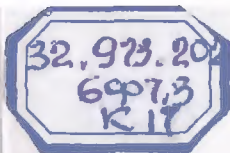


VISIT...

LANZAROTE
Caliente.COM



Основы

ВЫЧИСЛИТЕЛЬНЫХ
СИСТЕМ

Основные концепции

нейронных сетей



Роберт Каллан

Основные концепции

нейронных сетей

Р. Каллан

THE ESSENCE OF

Neural Networks

Robert Callan

Southampton Institute



Prentice Hall Europe

LONDON NEW YORK TORONTO SYDNEY TOKYO
SINGAPORE MADRID MEXICO CITY MUNICH PARIS

Основные концепции

нейронных сетей

Роберт Каллан

Саутгемптонский институт



Издательский дом "Вильямс"
Москва • Санкт-Петербург • Киев
2001

Б21792 32.973.202
697.5
к 17

ББК 32.973.26-018.2.75

К17

УДК 681.3.07

Издательский дом "Вильямс"

Перевод с английского и редакция А.Г. Сивака

По общим вопросам обращайтесь в Издательский дом "Вильямс"
по адресу: info@williamspublishing.com, http://www.williamspublishing.com



Каллан, Роберт.

К17 Основные концепции нейронных сетей. : Пер. с англ. — М. : Издательский дом "Вильямс", 2001. — с. : ил. — Парал. тит. англ.

ISBN 5-8459-0210-X (рус.)

Эта книга является первой в полном курсе по нейронным сетям. Ее целью является раскрытие основных понятий и изучение основных моделей нейронных сетей с глубиной, достаточной для того, чтобы опытный программист мог реализовать такую сеть на том языке программирования, который покажется ему предпочтительнее. В книге рассматриваются основные модели нейронных сетей, важные для понимания основ изучаемого предмета, и обсуждаются связи между нейронными сетями и традиционными понятиями из области искусственного интеллекта.

ББК 32.973.26-018.2.75

Все названия программных продуктов являются зарегистрированными торговыми марками соответствующих фирм.

Никакая часть настоящего издания ни в каких целях не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами, будь то электронные или механические, включая фотокопирование и запись на магнитный носитель, если на это нет письменного разрешения издательства Prentice Hall Europe.

Authorized translation from the English language edition published by Prentice Hall Europe, Copyright © 1999

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Russian language edition published by Williams Publishing House according to the Agreement with R&I Enterprises International, Copyright © 2001

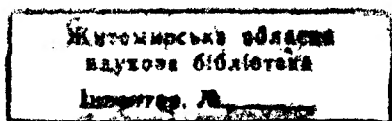
ISBN 5-8459-0210-X (рус.)

ISBN 0-13-908732-X (англ.)

999438

© Издательский дом "Вильямс", 2001

© Prentice Hall Europe, 1999



Оглавление

Предисловие	9
Благодарности	12
Глава 1. Введение	13
Глава 2. Классификация образцов	36
Глава 3. Кластеризация образцов	80
Глава 4. Ассоциация образцов	108
Глава 5. Рекуррентные сети	127
Глава 6. Другие модели сетей и практические вопросы	145
Глава 7. Связь с искусственным интеллектом	181
Глава 8. Синтез символов с помощью нейронных сетей	218
Предметный указатель	286

Содержание

Благодарности	12
Глава 1. Введение	13
1.1. Введение	13
1.2. Основные компоненты	15
1.3. Обучение нейронной сети	27
1.4. Простой пример обучения	28
1.4.1. Вывод уравнений для m и c	30
1.5. Резюме	32
1.6. Дополнительная литература	33
1.7. Упражнения	34
Глава 2. Классификация образцов	36
2.1. Приложения	36
2.2. Основные идеи	37
2.2.1. Функция выбора решения	37
2.2.2. Корректировка весов	41
2.2.3. Минимизация квадрата ошибки	43
2.3. Линейные и нелинейные проблемы	44
2.4. Обучение по алгоритму обратного распространения ошибок	51
2.4.1. Немного теории	53
2.4.2. Алгоритм обратного распространения ошибок	55
2.4.3. Практические рекомендации	60
2.5. Использование сети с обратным распространением ошибок	62
2.5.1. Классификация чисел	62
2.5.2. Классификация символов	63
2.5.3. Прогнозирование погоды	66
2.6. Сети с радиальными базисными функциями	67
2.7. Резюме	71
2.8. Дополнительная литература	77
2.9. Упражнения	77
Глава 3. Кластеризация образцов	80
3.1. Основные идеи	80
3.2. Самоорганизующаяся карта признаков	83
3.2.1. Алгоритм	87
3.2.2. Обучение сети SOFM	88
3.2.3. Дополнительные сведения о сети SOFM	97

3.3. Эксперимент	100
3.4. Резюме	102
3.5. Дополнительная литература	103
3.6. Упражнения	103
Глава 4. Ассоциация образцов	108
4.1. Введение	108
4.2. Дискретная сеть Хопфилда	109
4.2.1. Функция энергии	114
4.3. Двухнаправленная ассоциативная память	116
4.4. Автоассоциативное обратное распространение ошибок	121
4.5. Резюме	124
4.6. Дополнительная литература	125
4.7. Упражнения	125
Глава 5. Рекуррентные сети	127
5.1. Введение	127
5.2. Обратное распространение во времени	128
5.3. Простая рекуррентная сеть	133
5.3.1. Применение сети SRN	135
5.4. Резюме	143
5.5. Дополнительная литература	143
5.6. Упражнения	143
Глава 6. Другие модели сетей и практические вопросы	145
6.1. Введение	145
6.2. Сети, использующие статистический подход	146
6.2.1. Метод модельной “закалки”	146
6.2.2. Вероятностные нейронные сети	152
6.3. Пример модульной нейронной сети	164
6.4. Практические вопросы обучения нейронных сетей	168
6.4.1. Выбор модели сети	168
6.4.2. Архитектура	169
6.4.3. Данные	169
6.4.4. Локальные минимумы	175
6.4.5. Обобщение	176
6.5. Резюме	177
6.6. Дополнительная литература	178
6.7. Упражнения	178
Глава 7. Связь с искусственным интеллектом	181
7.1. Введение	181
7.2. Природа интеллекта	183

7.2.1. Знание и представление	183
7.2.2. Рассуждения	185
7.2.3. Обучение	187
7.3. Гипотеза символьных систем	188
7.3.1. Поиск	188
7.3.2. Продукционные системы	191
7.4. Представление с помощью символов	192
7.4.1. Исчисление высказываний	194
7.4.2. Исчисление предикатов	199
7.4.3. Другие символьные языки	200
7.4.4. Язык Prolog	201
7.5. Понимание речи	204
7.5.1. Синтаксический анализ	207
7.5.2. Семантический анализ	209
7.6. Символьные связи нейронных сетей	212
7.7. Резюме	214
7.8. Дополнительная литература	215
7.9. Упражнения	215
Глава 8. Синтез символов с помощью нейронных сетей	218
8.1. Нейронные сети в символьной форме	218
8.2. Рекурсивная автоассоциативная память	220
8.2.1. Обучение RAAM	224
8.3. Представления нейронных сетей	226
8.3.1. Локальные и распределенные представления	226
8.3.2. Пространственное сохранение структуры	228
8.3.3. Контекст	233
8.3.4. Символьное представление и представление нейронных сетей	235
8.4. Обработка речи	239
8.4.1. Синтаксический анализ	239
8.4.2. Преобразование предложений	241
8.4.3. Завершенные модели	242
8.5. Другие вопросы, касающиеся представлений	253
8.5.1. Обобщение	253
8.5.2. Проблема обоснования символов	256
8.6. Возможность машинного общения	259
8.7. Резюме	264
8.8. Дополнительная литература	265
8.9. Упражнения	266
Предметный указатель	286

Предисловие

Задумайтесь над некоторыми ежедневно выполняемыми задачами. Вы сидите за столом в офисе, а в это время в комнату входит в новой шляпе ваш коллега — мужчина, выглядящий немного помолодевшим от того, что он сбрил бороду. Узнаете ли вы его? Несомненно, поскольку маскировка его целью не является. Он спрашивает вас: “Где книга, которую вы взяли у меня вчера?”. Вы помните о книге и интерпретируете вопрос как просьбу вернуть книгу. Вы переводите взгляд на свой стол и видите на ящике с дискетами стопку деловых бумаг, среди которых лежит и та книга, о которой идет речь. Вы протягиваете руку к книге, не задумываясь о движениях, которые при этом должна выполнить ваша рука, извлекаете книгу из стопки с документами и отдаете ее своему коллеге.

Эти ежедневно выполняемые задачи не требуют особых усилий, но каждая из них включает множество точно рассчитанных шагов. Мы должны были бы с благоговением взирать на машину, работающую так же совершенно. Масштабы проблемы создания такой машины можно почувствовать, пытаясь программировать компьютерную систему для распознавания объектов по внешнему виду или другим признакам, в зависимости от контекста, когда приходится анализировать выполняемые действия, планировать движения робота и т.п. Пытаясь решить такие сложные задачи, многие ученые обратили внимание на машины, сходные по принципу работы с нашим собственным компьютером — человеческим мозгом. Такие машины, использующие сети, состоящие из простых обрабатывающих элементов и легко адаптируемые к выполнению совершенно разных задач, называются *нейронными сетями*. Нейронные сети не программируются, а обучаются тому, как правильно реализовать конкретную задачу.

Нейронные сети оказались очень полезными при решении задач распознавания образов типа идентификации подводных объектов по сигналам гидролокатора или выявления фальшивых кредитных карточек. Число коммерческих приложений, использующих нейронные сети, постоянно растет. Развивается и теоретическая база применения нейронных сетей в так называемых когнитивных задачах типа задач понимания обычного разговорного языка или задач управления движением автономного транспортного средства. Такие задачи высокого уровня сложности

долгие годы были предметом изучения дисциплины, областью интересов которой является искусственный интеллект. Теперь выделяют новую область искусственного интеллекта, в рамках которой предпринимаются попытки объединить идеи “традиционного искусственного интеллекта” и идеи теории нейронных сетей. Этот “новый искусственный интеллект” несет в себе большие потенциальные возможности для перехода компьютерных систем на следующий уровень развития.

Данная книга предлагается в качестве первой книги по теории нейронных сетей для студентов старших курсов. Большинство книг по данной тематике предполагает наличие у читателя хорошей математической подготовки. Полностью избежать использования математики при изучении нейронных сетей невозможно, но в данной книге сделана попытка свести использование математического аппарата к минимуму — здесь, в основном, математика используется для сжатого представления алгоритмов. Причем если используемые в книге математические выкладки для кого-то окажутся слишком сложными, читателю предоставлена возможность разобрать имеющиеся примеры, чтобы увидеть, как соответствующие обозначения используются в примерах. Главной целью книги является объяснение базовых концепций теории нейронных сетей, но в ней достаточно подробно рассматриваются и наиболее важные модели нейронных сетей, чтобы опытный программист мог реализовать такую сеть на том языке программирования, который окажется для него предпочтительнее.

В первых шести главах описываются наиболее важные модели нейронных сетей, что должно обеспечить понимание основ изучаемого предмета. Эти шесть глав занимают чуть более двух третей книги. У нас возникало искушение расширить использование математического аппарата и алгоритмов в этих главах, но это могло бы сузить круг читателей книги. Поэтому представление материала в некоторых частях книги было сознательно сжато, а объем разделов с примерами — увеличен. Например, алгоритм обратного распространения ошибок, о котором идет речь в главе 2, из представленных в книге алгоритмов является, вероятно, наиболее трудным для понимания, но вместе с тем этот алгоритм оказывается очень простым, если рассмотреть примеры его реализации на практике. По этой причине в дополнение к некоторым простым примерам в конце глав предлагаются более сложные. Читатель может также обратиться к приложению А, где в краткой форме излагаются необходимые сведения из области линейной алгебры, а также разъясняются некоторые обозначения, используемые в главе 2. В последних двух главах книги обсуждаются некоторые связи между нейронными сетями и понятиями из

сферы традиционного искусственного интеллекта. Глава 7 содержит подготовительный материал для главы 8. Сегодня наблюдается рост интереса к идее синтеза нейронных сетей и традиционного искусственного интеллекта. В отличие от первых шести глав, в главе 8 не ставится цель предложить читателю знания, которые могут сразу же найти непосредственное применение. Некоторые из предлагаемых там идей могут показаться довольно трудными для понимания и слишком абстрактными. Но соответствующий материал дает возможность представить решения, в которых с помощью использования модульных сетевых систем решаются некоторые весьма сложные вычислительные проблемы. По целому ряду направлений разработка новых моделей нейронных сетей несомненно будет продолжаться, но читатель уже получит достаточно весомый объем знаний, позволяющих решать задачи, относящиеся к сфере традиционного искусственного интеллекта. Предполагается, что у читателя будет возникать необходимость время от времени снова обращаться к главе 8 за информацией, но обратите внимание и на публикации, предлагаемые в списке литературы в конце книги.

В дополнение к материалу данной книги предлагается Web-страница, размещенная по адресу

<http://www.solent.ac.uk/syseng/faculty/html/staff/rcallan/essnn>,

где можно найти и программное обеспечение по нейронным сетям, которое читатель может использовать на своем компьютере.

Р. Каллан.

Благодарности

Я хотел бы поблагодарить Доминик Палмер-Браун (Dominic Palmer-Brown) за ряд полезных замечаний, касающихся глав 7 и 8. Благодарю за помощь Джона Флаккетта (John Flackett), особенно за предложенную им схему рекурсивной автоассоциативной памяти, представленную в главе 8. Благодарю Дейва Парсонса (Dave Parsons) за его полезные комментарии к первым двум главам. Я также благодарю за поддержку Хешама Аззама (Hesham Azzam), с которым я работаю уже много лет. Наконец, я хотел бы поблагодарить рецензентов за их ценные рекомендации, позволившие улучшить окончательный вариант текста книги по сравнению с исходным, и Джеки Харбор (Jackie Harbor) из Prentice Hall за помощь и поддержку.

Глава 1

Введение

Задача. Определение основных элементов нейронной сети.

Цели. Вы должны понять:

как объяснить простыми словами, что такое нейронная сеть;
что такое элемент, его связи и функция активности;
как представить связи в сети в матричной форме;
как объяснить простыми словами, что означает обучение сети.

1.1. Введение

Искусственные нейронные сети представляют собой устройства параллельных вычислений, состоящие из множества взаимодействующих простых процессоров. Такие процессоры обычно исключительно просты, особенно в сравнении с процессорами, используемыми в персональных компьютерах. Каждый процессор подобной сети имеет дело только с сигналами, которые он периодически получает, и сигналами, которые он периодически посылает другим процессорам, и, тем не менее, будучи соединенными в достаточно большую сеть с управляемым взаимодействием, такие локально простые процессоры вместе способны выполнять довольно сложные задачи.

Разработка искусственных нейронных сетей началась еще на заре XX столетия, но только в 90-х годах, когда были преодолены некоторые теоретические барьеры, а вычислительные системы стали достаточно мощными, нейронные сети получили широкое признание. Слово “искусственные” в данном контексте иногда используется для того, чтобы подчеркнуть, что речь идет об искусственном устройстве, а не о реальных биологических нейронных системах типа той, которую имеет человек. Создание искусственных нейронных сетей было инспирировано попытками понять принципы работы человеческого мозга и, без сомнения, это будет влиять и на дальнейшее их развитие. Однако, в сравнении с человеческим мозгом, искусственные нейронные сети сегодня представляют собой весьма упрощенные абстракции. Когда ясно, в каком

контексте обсуждаются эти сети, слово “искусственный” обычно опускают. Кроме того, когда требуется подчеркнуть вычислительные возможности, а не биологическое соответствие, искусственные нейронные сети называют коннекциями. При этом целью “коннекционистов” является наделение нейронной сети возможностью решать конкретные задачи, а не имитировать с максимальной точностью биологический процесс.

Хотя нейронные сети могут быть реализованы в виде быстрых аппаратных устройств (и такие реализации действительно существуют), большинство исследований выполняется с использованием программного моделирования на обычных компьютерах. Программное моделирование обеспечивает достаточно дешевую и гибкую среду для поиска и проверки исследовательских идей, а для многих реальных приложений такое моделирование оказывается вполне адекватным и достаточным. Например, программная реализация нейронной сети может использоваться для составления плана кредитных выплат индивидуума, обращающегося в банк за займом. Хотя решение на основе нейронной сети может выглядеть и вести себя как обычное программное обеспечение, они различны в принципе, поскольку большинство реализаций на основе нейронных сетей “обучается”, а не программируется: сеть учится выполнять задачу, а не программируется непосредственно. На самом деле в большинстве случаев нейронные сети используются тогда, когда невозможно написать подходящую программу, или по причине того, что найденное нейронной сетью решение оказывается более совершенным. Например, будучи экспертом по продаже недвижимости, вы можете из своего опыта прекрасно знать, какие факторы влияют на продажную цену каждого конкретного дома, но при этом часто имеются такие особенности, которые будет весьма трудно объяснить программисту. Агентство по продаже недвижимости может пожелать иметь “предсказателя цен на основе нейронной сети”, обученного на множестве примеров реальных продаж тому, какие факторы влияют на цену продаваемого объекта. и тому, какую относительную важность имеет каждый из этих факторов. Но здесь более важным оказывается то, что решение на основе нейронной сети является более гибким, поскольку соответствующая система может в дальнейшем совершенствовать точность предсказаний по мере накопления ею опыта и адаптироваться к происходящим на рынке изменениям.

Решения на основе нейронных сетей становятся все более совершенными и, несомненно, в будущем наши возможности по разработке соответствующих устройств возрастут за счет лучшего понимания их основополагающих принципов. Но уже сегодня имеется немало впечатляющих разработок. База приложений нейронных сетей просто огромна: выявление

фальшивых кредитных карточек, прогнозирование изменений на фондовой бирже, составление кредитных планов, оптическое распознавание символов, профилактика и диагностика заболеваний человека, наблюдение за техническим состоянием машин и механизмов, автоматическое управление движением автомобиля, принятие решений при посадке поврежденного летательного аппарата и т.д. Дальнейшие успехи в разработке искусственных нейронных сетей будут зависеть от дальнейшего понимания принципов работы человеческого мозга, но здесь имеется и обратная связь: искусственные нейронные сети являются одним из средств, с помощью которых совершенствуется наше представление о процессах, происходящих в нервной системе человека, выступая в качестве моделей соответствующих процессов.

Будущее нейронных сетей кажется вполне ясным, и сегодня это та область знаний, о которой должны иметь определенное представление все научные специалисты, работающие в области компьютерных технологий, равно как и многие инженеры и научные работники смежных специальностей.

1.2. Основные компоненты

Нейронная сеть является совокупностью *элементов*, соединенных некоторым образом так, чтобы между ними обеспечивалось взаимодействие. Эти элементы, называемые также *нейронами* или *узлами*, представляют собой простые процессоры, вычислительные возможности которых обычно ограничиваются некоторым правилом комбинирования входных сигналов и правилом активизации, позволяющим вычислить выходной сигнал по совокупности входных сигналов. Выходной сигнал элемента может посылаться другим элементам по взвешенным *связям*, с каждой из которых связан *весовой коэффициент* или *вес*. В зависимости от значения весового коэффициента передаваемый сигнал или усиливается, или подавляется. Элемент нейронной сети схематически показан на рис. 1.1.

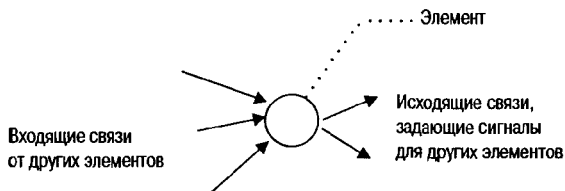


Рис. 1.1. Отдельный элемент сети

Один из самых привлекательных аспектов использования нейронных сетей заключается в том, что, хотя элементы такой сети имеют очень ограниченные вычислительные возможности, вся сеть в целом, объединяя большое число таких элементов, оказывается способной выполнять довольно сложные задачи (рис. 1.2).

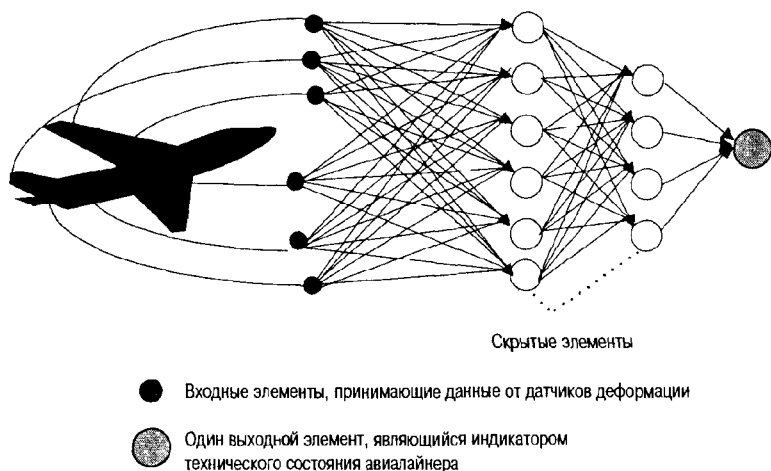


Рис. 1.2. Схема применения нейронной сети для контроля технического состояния авиалайнера. Связи элементов показаны стрелками. Входные элементы получают информацию непосредственно от датчиков, установленных на борту самолета. Выходной элемент является индикатором технического состояния летательного аппарата

Структура связей отражает детали конструкции сети, а именно то, какие элементы соединены, в каком направлении работают соединения и каков уровень значимости (т.е. вес) каждого из соединений. Задача, которую понимает сеть (или ее программа), описывается в терминах весовых значений связей, связывающих элементы. Структура связей обычно определяется в два этапа: сначала разработчик системы указывает, какие элементы должны быть связаны и в каком направлении, а затем в процессе фазы обучения определяются значения соответствующих весовых коэффициентов.

Весовые коэффициенты можно определить и без проведения обучения, но как раз самое большое преимущество нейронных сетей заключается в их способности обучаться выполнению задачи на основе тех данных, которые сеть будет получать в процессе реальной работы. Для многих приложений обучение является не только средством программирования сети, когда нет достаточных знаний о способе решения задачи,

позволяющих выполнить программирование в традиционной форме, но часто единственной целью обучения является проверка того, что сеть действительно сможет научиться решать поставленные перед ней задачи.

Существует множество различных типов нейронных сетей, но все рассматриваемые в данной книге сети обладают рядом общих характеристик, которые можно представить с помощью следующих абстракций.

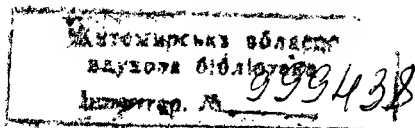
- Множество простых процессоров
- Структура связей
- Правило распространения сигналов в сети
- Правило комбинирования входящих сигналов
- Правило вычисления сигнала активности
- Правило обучения, корректирующее связи

Множество простых процессоров

С каждым процессором (т.е. обрабатывающим элементом сети) связывается набор входящих связей, по которым к данному элементу поступают сигналы от других элементов сети, и набор исходящих связей, по которым сигналы данного элемента передаются другим элементам. Некоторые элементы предназначены для получения сигналов из внешней среды (и поэтому называются *входными элементами*), а некоторые — для вывода во внешнюю среду результатов вычислений (и поэтому такие элементы сети называются *выходными элементами*). Любая вычислительная машина имеет хотя бы одно устройство ввода (например, клавиатуру), с помощью которого система получает данные из внешней среды, и устройство вывода (например, монитор), с помощью которого отображаются результаты вычислений. В случае программного моделирования реальных процессов на входные элементы обычно подаются уже предварительно подготовленные данные из некоторого файла данных, а не от непосредственно связанных с внешней средой датчиков.

Структура связей

Структура связей отражает то, как соединены элементы сети. В одной модели (т.е. для одного типа сетей) каждый элемент может быть связан со всеми другими элементами сети, в другой модели элементы могут быть организованы в некоторой упорядоченной по уровням (слоям) иерархии, где связи допускаются только между элементами в смежных слоях, а в третьей — могут допускаться обратные связи между смежными слоями или внутри одного слоя, или же допускаться посылка сигналов элементами самим себе. Возможности здесь практически бесконечны, но обычно для каждой конкретной модели сети указывается тип допустимых связей. Каждая связь определяется



тремя параметрами: элементом, от которого исходит данная связь, элементом, к которому данная связь направлена, и числом (обычно действительным), указывающим весовой коэффициент (т.е. вес связи). Отрицательное значение веса соответствует подавлению активности соответствующего элемента, а положительное значение — усилению его активности. Абсолютное значение весового коэффициента характеризует силу связи.

Структура связей обычно представляется в виде весовой матрицы W , в которой каждый элемент w_{ij} представляет величину весового коэффициента для связи, идущей от элемента i к элементу j (обратите внимание на то, что во многих публикациях связи в весовых матрицах предполагаются идущими от элемента j к элементу i , что, очевидно, следует учитывать при представлении матричных и векторных операций). Для описания структуры связей может использоваться не одна, а несколько весовых матриц, если элементы сети оказываются сгруппированными в слои. На рис. 1.3 и 1.4 предлагаются примеры представления структуры связей в виде соответствующих матриц.

Матрица весов является памятью сети, хранящей информацию о том, как должна выполняться задача.

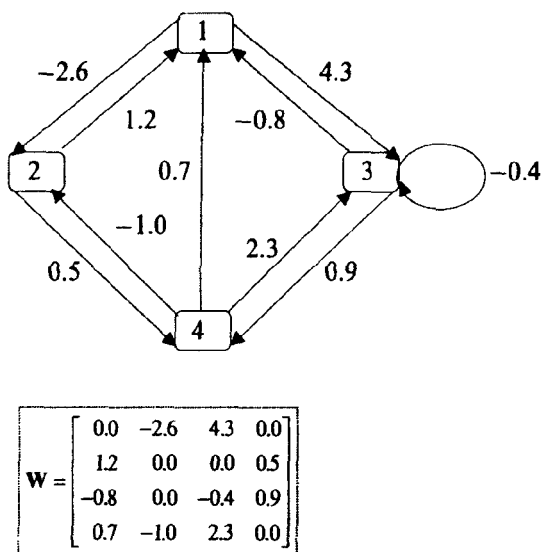


Рис. 1.3. Матрица, описывающая сетевые связи. В данном случае, например, вес связи элемента 3 (строка 3) с элементом 1 (столбец 1) обозначается символом $w_{31} = -0.8$

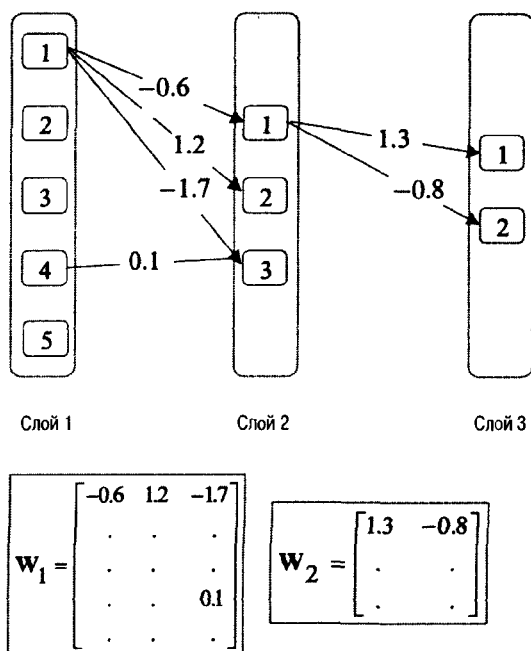


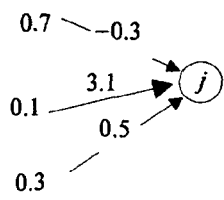
Рис. 1.4. Матрицы, описывающие сетевые связи. В данном случае для каждого слоя имеется своя отдельная матрица

Правило распространения сигналов в сети

В обычных компьютерных программах используются условия, выполнение которых определяет начало и конец различных процессов. То же самое верно и для нейронных сетей. Каждая конкретная модель сети предполагает наличие некоторого правила обновления состояния элементов сети (т.е. правила комбинирования входящих сигналов и вычисления исходящего сигнала) и посылки сигнала другим элементам. При этом в одних моделях моменты обновления элементов выбираются случайным образом, в других же моделях обновление некоторых групп элементов допускается только после обновления определенных групп других элементов.

Правило комбинирования входящих сигналов

Довольно часто входящие сигналы элемента предполагается комбинировать путем суммирования их взвешенных значений. Пример этого метода суммирования показан на рис. 1.5, где net_j обозначает результат



$$net_j = \sum_{i=1}^n x_i w_{ij}$$

$$net_j = (0.7 \times -0.3) + (0.1 \times 3.1) + (0.3 \times 0.5)$$

$$= 0.25$$

или в векторном представлении

$$[0.7 \quad 0.1 \quad 0.3] \begin{bmatrix} -0.3 \\ 3.1 \\ 0.5 \end{bmatrix}$$

Рис. 1.5. Типичный метод суммирования сигналов, направленных конкретному элементу

комбинирования вклада элемента j , x_i — выход элемента i , а n — число задействованных связей. Используются и другие формы комбинирования входящих сигналов, и другим часто встречающимся методом является рассмотрение квадрата разности между значением силы связи и значением передаваемого по связи сигнала с последующим суммированием таких разностей для всех входящих связей данного элемента.

Правило вычисления сигнала активности

Для всех элементов имеется правило вычисления выходного значения, которое предполагается передать другим элементам или во внешнюю среду (если речь идет о выходном элементе, представляющем конечный результат вычислений). Это правило называют *функцией активности*, а соответствующее выходное значение называют *активностью* соответствующего элемента. Активность может представляться либо некоторым действительным значением произвольного вида, либо действительным значением из некоторого ограниченного интервала значений (например, из интервала $[0, 1]$), или же некоторым значением из определенного дискретного набора значений (например, $\{0, 1\}$ или $\{+1, -1\}$). На вход функции активности поступает значение комбинированного вклада данного элемента. Примеры функций активности приводятся ниже.

Тождественная функция

Функция активности для входных элементов может быть тождественной функцией, и это просто означает, что значение активности (сигнал, посылаемый другим элементам) оказывается в точности равным комбинированному вводу (рис. 1.6). Входные элементы обычно предназначены для распределения вводимых сигналов между другими элементами сети, поэтому для входных элементов обычно требуется, чтобы исходящий от элемента сигнал был таким же, как и входящий. В отличие от других элементов сети, входные элементы имеют только по одному входному значению. Например, каждый входной элемент может получать сигнал от одного соответствующего ему датчика, размещенного на фюзеляже самолета. Один этот элемент связывается со многими другими элементами сети, так что данные, полученные от одного датчика, оказываются распределенными между многими элементами сети. Поскольку входные элементы предназначены исключительно для того, чтобы распределять сигналы, получаемые из внешней среды, многие исследователи вообще не считают входные элементы частью нейронной сети.

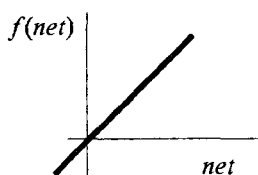


Рис. 1.6. Здесь активность в точности равна комбинированному вводу. Обратите внимание на то, что активность обозначается символом $f(net)$

Пороговая функция

В большинстве моделей нейронных сетей используются нелинейные функции активности. Пороговая функция ограничивает активность значениями 1 или 0 в зависимости от значения комбинированного ввода в сравнении с некоторой пороговой величиной θ (рис. 1.7).

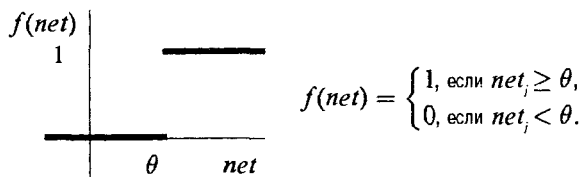


Рис. 1.7. Пороговая функция

Чаще всего удобнее вычесть пороговое значение (называемое *смещением* или *сдвигом*) из значения комбинированного ввода и рассмотреть пороговую функцию в ее математически эквивалентной форме, показанной на рис. 1.8. Сдвиг w_0 в данном случае оказывается отрицательным, а значение комбинированного ввода вычисляется по формуле

$$net_j = w_0 + \sum_{i=1}^n x_i w_{ij}.$$

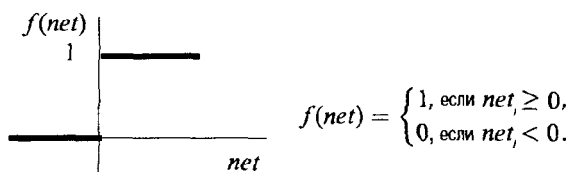


Рис. 1.8. Пороговая функция с учтенным смещением

Сдвиг обычно интерпретируется как связь, исходящая от элемента, активность которого всегда равна 1 (рис. 1.9). Комбинированный ввод в данном случае можно представить в виде

$$net_j = \sum_{i=0}^n x_i w_{ij},$$

где x_0 всегда считается равным 1.

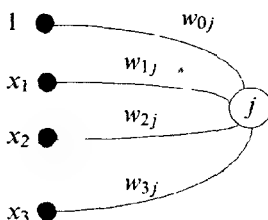


Рис. 1.9. Для удобства компонент смещения часто интерпретируется как связь с элементом предыдущего слоя в предположении, что активность этого элемента всегда равна 1

Сигмоидальная функция

Наиболее часто используемой функцией активности является сигмоидальная функция. Выходные значения такой функции непрерывно заполняют диапазон от 0 до 1. Примером может служить логистическая функция, показанная на рис. 1.10:

$$f(net) = \frac{1}{1 + \exp(-net)}$$

Наклон и область выходных значений логистической функции могут быть разными. Например, для биполярного сигмоида областью выходных значений является диапазон между -1 и 1 .

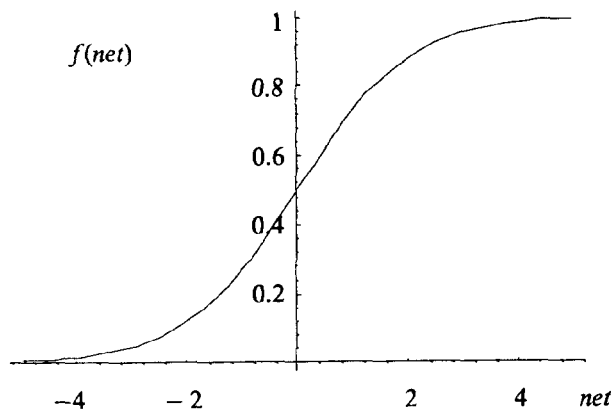


Рис. 1.10. Сигмоидальная функция

Пример 1.1

Этот пример иллюстрирует некоторые понятия, обсуждавшиеся выше. Предполагается, что рассматриваемая здесь сеть понимает отношение XOR. Отношение XOR отображает пару двоичных входных значений в 0 или 1, точное определение представлено в табл. 1.1.

Таблица 1.1. Определение XOR

Ввод		Вывод
x_1	x_2	
1	1	0
1	0	1
0	1	1
0	0	0

Модель сети показана на рис. 1.11, и в данном случае это сеть с прямой связью, в которой имеются два входных элемента, два скрытых элемента и один выходной элемент. *Прямая связь* означает, что все связи могут идти только в направлении от входного слоя к выходному. *Скрытые* элементы называются так потому, что они не получают данных от внешней среды непосредственно и не посылают данные непосредственно во внешнюю среду.

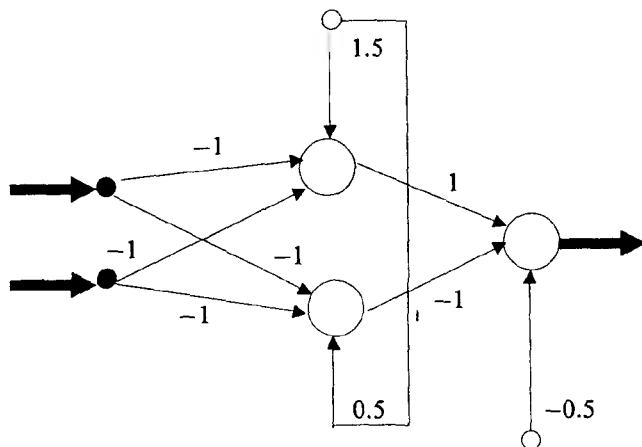


Рис. 1.11. Сеть для примера 1.1

В данном случае в роли внешней среды можем выступать мы сами, подавая различные значения на вход сети (т.е. входным элементам) и наблюдая результаты, полученные на ее выходе (т.е. на выходных элементах). Элементы сети разделены по слоям: входной слой содержит входные элементы, скрытый слой — скрытые элементы, а выходной — выходные. Число элементов каждого слоя зависит от решаемой проблемы, и мы обсудим это в главе 2, где сети с прямой связью будут рассмотрены подробнее. Пока же мы просто обращаем внимание на то, что число входных элементов равно числу вводимых в структуру значений, а число выходных — числу значений, подаваемых данной структурой на выход. В нашем случае значение комбинированного ввода вычисляется по формуле

$$net_j = \sum_{i=0}^n x_i w_{ij} ,$$

а вывод получается как результат применения пороговой функции

$$f(net) = \begin{cases} 1, & \text{если } net \geq 0, \\ 0, & \text{если } net < 0. \end{cases}$$

Вспомните, что для элементов входного слоя активность представляется значением, совпадающим со значением комбинированного ввода. Сигналы распространяются по сети от входного слоя к выходному, так что для каждого конкретного набора вводимых значений последовательность их обработки будет следующей:

входной слой → скрытый слой → выходной слой.

В качестве вводимых данных рассмотрим первую пару значений ввода из табл. 1.1, а именно пару значений [1, 1]. Для первого скрытого элемента со смещением 1.5 получаем

$$\begin{aligned} net &= (x_0 \times 1.5) + (x_1 \times -1) + (x_2 \times -1) \\ &= (1 \times 1.5) + (1 \times -1) + (1 \times -1) = -0.5, \end{aligned}$$

поэтому выходным значением элемента будет 0. Для второго скрытого элемента со смещением 0.5 получаем

$$\begin{aligned} net &= (x_0 \times 0.5) + (x_1 \times -1) + (x_2 \times -1) \\ &= (1 \times 0.5) + (1 \times -1) + (1 \times -1) = -1.5, \end{aligned}$$

поэтому выходным значением элемента тоже будет 0. Для выходного элемента со смещением -0.5 получаем

$$\begin{aligned} net &= (x_0 \times -0.5) + (x_1 \times 1) + (x_2 \times -1) \\ &= (1 \times -0.5) + (0 \times -1) + (0 \times -1) = -0.5, \end{aligned}$$

поэтому выходным значением будет 0. Если процедуру повторить для трех оставшихся пар, то мы увидим, что вывод указанной сети соответствует данным из последнего столбца табл. 1.1.

Правило обучения, корректирующее связи

Одно из главных преимуществ нейронных сетей заключается в том, что они предполагают наличие правил, с помощью которых сеть может программироваться автоматически. Например, можно рассмотреть следующую функцию, реализующую вышеприведенное определение XOR.

```
int XOR(int val_1, int val_2)
{
    if (val_1 == 1 && val_2 == 1)
        return 0;
    if (value_1 == 0 && val_2 == 0)
        return 0;
```

```

if (val_1 == 1 && val_2 == 0)
    return 1;
if (val_1 == 0 && val_2 == 1)
    return 1;
}

```

Нельзя сказать, что данный код оптимален и что данная функция не может быть реализована иначе. Мы уже видели, что сеть из примера 1.1 выполняет ту же задачу. Правильное выполнение операции XOR зависит от размещения элементов, выбора функции активности и набора весовых значений. Размещение элементов обычно фиксировано уже в начале обучения и точно так же оказывается заданной функция активности. Поэтому целью обучения является изменение весовых значений таким образом, чтобы в результате получить требуемые характеристики поведения сети.

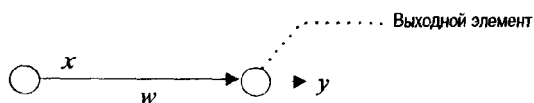


Рис. 1.12. Взвешенная связь двух элементов. Сигнал x умножается на весовой коэффициент w . Для выходного элемента функция активности является тождественной функцией, что означает равенство вывода y взвешенному вводу

Типичной формой обучения является управляемое обучение, когда для каждого набора данных, подающегося в процессе обучения на вход сети, соответствующий выходной набор известен. Обычно в начале обучения весовые коэффициенты устанавливаются равными случайным малым значениям, так что в первый раз при предъявлении сети учебного образца оказывается весьма маловероятным, чтобы сеть произвела верный вывод. Расхождение между тем, что даст сеть, и тем, что для данного учебного набора должно быть получено на самом деле, составляет ошибку, которая может использоваться для корректировки весов. Примером правила коррекции ошибок является дельта-правило, называемое также правилом Видроу–Хоффа (Widrow–Hoff rule). Взгляните на рис. 1.12, где выходной элемент имеет активность (т.е. вывод) y , а истинный вывод должен быть равным t . Ошибка δ задается следующей формулой:

$$\delta = t - y.$$

Сигнал, приходящий к выходному элементу, обозначен через x . В соответствии с дельта-правилом, необходимо внести коррекцию Δw , вычисляемую по формуле

$$\Delta w = \eta \delta x ,$$

где η обозначает действительное число, называемое *нормой обучения*. Новый весовой коэффициент устанавливается равным сумме значений старого веса и коррекции:

$$w = w + \Delta w .$$

В алгоритме обучения выполнению операций типа XOR, который будет представлен в главе 2, используется обобщенная версия дельта-правила.

В начале обучения весовые коэффициенты устанавливаются равными малым случайным значениям; например, из диапазона $[-0.3, +0.3]$. В процессе обучения на вход сети подаются образец за образцом, и в результате их обработки весовые коэффициенты корректируются до тех пор, пока для всех вводимых образцов ошибки не станут меньше некоторого приемлемого достаточно малого значения. В завершение процесса сеть тестируется на данных, не представленных в фазе обучения: в результате можно оценить, насколько хорошо сеть работает с данными, которые в процессе обучения были ей неизвестны.

1.3. Обучение нейронной сети

Качество работы нейронной сети сильно зависит от предъявляемого ей в процессе обучения набора учебных данных. Учебные данные должны быть типичными для задачи, решению которой обучается сеть. Обучение часто оказывается уникальным процессом, когда приемлемые решения многих проблем могут быть получены только в процессе многочисленных экспериментов. Разработчикам решения на основе нейронной сети требуется следующее.

- Выбрать соответствующую модель сети
- Определить топологию сети (т.е. число элементов и их связи)
- Указать параметры обучения

Часто разработчику необходимо выполнить и предварительную подготовку данных. Такая предварительная подготовка может быть совсем простой, — например, перевод с помощью масштабирования значений всех признаков (т.е. переменных) в диапазон от 0 до 1, — а может включать использование и более сложных статистических процедур. Однако здесь следует подчеркнуть, что долгосрочной целью разработки нейронных сетей является минимизация необходимости прямого влияния разработчика на процесс нахождения решения, так как главным преимуществом нейронных сетей является их потенциальная возможность вырабатывать

собственные решения. На практике лучшие результаты получаются тогда, когда имеется четкое понимание рассматриваемой проблемной области знаний и концептуальное понимание проблем построения нейронной сети.

Данные, используемые для обучения нейронной сети, обычно разделяются на две категории: одни данные используются для обучения, а другие — для тестирования. На самом деле реальные качества нейронной сети выявляются только во время тестирования, поскольку успешное завершение обучения сети должно означать отсутствие признаков неправильной работы сети во время ее тестирования. Процесс тестирования разрабатывается так, чтобы в его ходе для данной сети можно было бы оценить ее способность обобщать полученные знания. Обобщение в данном случае означает способность сети правильно выполнять задачу с данными, которые оказываются хотя и аналогичными данным, предъявлявшимся сети в процессе обучения, но все же отличными от них.

1.4. Простой пример обучения

Давайте рассмотрим относительно простую задачу и выясним, как для ее решения можно использовать самый простой тип нейронной сети. Наша сеть будет состоять только из одного входного и одного выходного элементов.

Многие из нас в школе на уроке физики выполняли эксперимент, в котором требовалось измерить отклонение металлической пластины в зависимости от приложенной к ней нагрузки — гирь определенного веса. Потом необходимо было начертить график отклонения в зависимости от веса приложенной нагрузки. График строился по точкам, которые должны были в идеале образовать прямую линию. Используя метод *наименьших квадратов*, по полученным точкам можно построить прямую линию, с помощью которой можно будет предсказать, насколько отклонится пластина под нагрузкой, для которой отклонение не измерялось. Многие проблемы моделируются с помощью построения прямых (или кривых) линий по имеющимся данным. Например, можно анализировать графики производства за прошедшие несколько лет, чтобы оценить, сколько стиральных машин изготовит фирма в течение следующих двух лет. Если тенденция изменения данных соответствует прямой линии, мы получим нечто похожее на диаграмму, показанную на рис. 1.13, где данные располагаются вдоль прямой, хотя и не существует прямой линии, на которой они лежали бы в точности. Этого и следовало ожидать — в реальном мире при любом измерении всегда имеются ошибки.

Уравнение прямой задается формулой

$$y = mx + c,$$

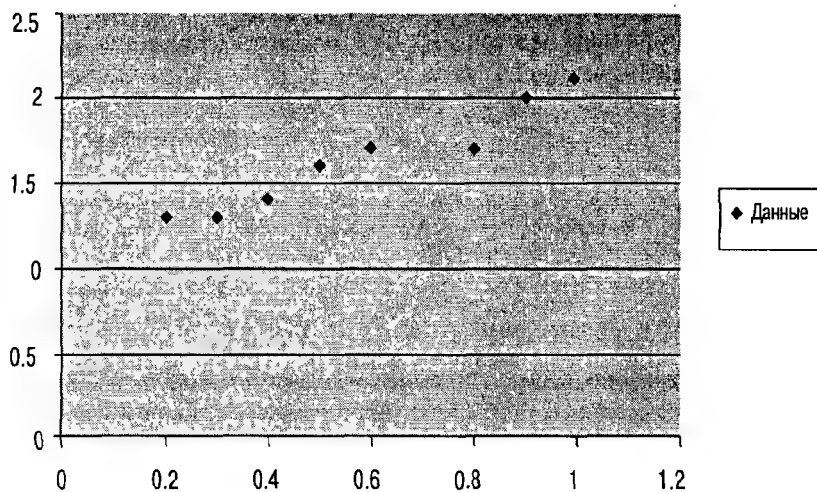


Рис. 1.13. Данные располагаются вдоль прямой линии, но не лежат в точности на прямой из-за ошибок измерения

где y и x являются переменными (например, отклонением и нагрузкой), m определяет наклон или градиент прямой, а c — значение сдвига (т.е. точку, в которой прямая пересекает ось y). Можно провести прямую на глаз, а затем измерить ее наклон и значение сдвига, но метод наименьших квадратов дает нам возможность вычислить m и c . Что означает “наиболее подходящая прямая”? В данном случае это прямая, для которой сумма квадратов ошибок для всех точек, соответствующих имеющимся данным, оказывается наименьшей. Что такое ошибки для набора точек, показано на рис. 1.14. Чтобы найти сумму квадратов ошибок, следует возвести значение каждой из ошибок в квадрат и просуммировать все полученные таким образом значения.

При использовании метода наименьших квадратов m находится по формуле

$$m = \frac{n \sum x_i y_i - \sum x_i \sum y_i}{n \sum x_i^2 - (\sum x_i)^2}, \quad (1.1)$$

а c — по формуле

$$c = \frac{\sum y_i - m \sum x_i}{n}, \quad (1.2)$$

где x_i и y_i представляют значения координат для точки i , а n равно числу точек. Суммирование выполняется по всем точкам данных.



Рис. 1.14. Каждой точке соответствует своя ошибка, равная расстоянию от точки до прямой

1.4.1. Вывод уравнений для m и c

Этот подраздел не является обязательным, но может оказаться полезным для лучшего понимания некоторых теоретических понятий, о которых пойдет речь в главе 2.

Как только с данными оказывается сопоставленной некоторая прямая, с помощью уравнения этой прямой для любого данного значения x можно получить оценку соответствующего значения y . В реальности любой оценке значения y соответствует своя ошибка. Поэтому для оценки y мы можем записать:

$$y_i = mx_i + c + e_i, \quad (1.3)$$

где e_i обозначает ошибку для точки i . Сумма квадратов ошибок, E , вычисляется по формуле

$$E = \sum e_i^2. \quad (1.4)$$

Используя равенства (1.3) и (1.4), получаем:

$$E = \sum [y_i - (mx_i + c)]^2. \quad (1.5)$$

Теперь, рассмотрев частные производные уравнения (1.5), мы увидим, как зависит общая ошибка от изменения m и c :

$$\frac{\partial E}{\partial m} = -2 \sum x_i [y_i - (mx_i + c)], \quad (1.6)$$

$$\frac{\partial E}{\partial c} = -2 \sum [y_i - (mx_i + c)]. \quad (1.7)$$

Если (1.6) и (1.7) приравнять к 0 и решить соответствующие уравнения, будут получены равенства (1.1) и (1.2).

Метод наименьших квадратов дает эффективный способ нахождения прямой, наилучшим образом соответствующей имеющемуся набору дан-

ных. Метод прост в применении, но его обоснование требует определенного уровня знания математики. Для нахождения прямой, соответствующей имеющемуся набору данных, можно использовать нейронную сеть. Такой сети нужно просто предоставить учебные данные и дать возможность обучиться на них.

Сеть с одним входным и одним выходным элементами была обучена проводить прямую линию на основе анализа имеющихся данных (см. рис. 1.13). В этой сети использовалась линейная функция активности. Задача требовала, чтобы сеть оценивала m и c , поэтому m и c являются параметрами сети (т.е. весовыми коэффициентами), значения для которых перед началом обучения были выбраны случайным образом из диапазона между -0.3 и $+0.3$. Модель сети показана на рис. 1.15. Данными для обучения были значения координаты x для каждой точки и соответствующие целевые значения координаты y . Вес c на входе имеет значение 1 (c , умноженное на 1, равно c ; этот вес задает смещение). Для обучения сети использовалось дельта-правило, а норма обучения была выбрана равной 0.1. Обучение заканчивалось после рассмотрения каждой точки 10000 раз.

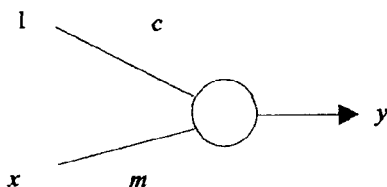


Рис. 1.15. Линейный элемент, который можно обучить найти прямую для данных на рис. 1.16

Оценки для m и c , полученные с помощью сети, в результате применения метода наименьших квадратов, представлены в следующей таблице, а линия, найденная сетью, показана на рис.1.16.

Параметр	Метод наименьших квадратов	Сеть
m	1.0085	1.0284
c	1.0450	1.0360

На самом деле совсем не удивительно, что сеть по сравнению с методом наименьших квадратов дает сравнимые результаты, поскольку,

как мы увидим в главе 2, дельта-правило обучения выводится из принципа минимизации суммы квадратов ошибок. Наша сеть решила данную задачу путем обобщения результатов вывода в зависимости от ввода, подобрав для них некоторую линейную зависимость. В нашем случае сеть моделирует прямую линию, но большие нейронные сети с нелинейными функциями активности могут подбирать для данных обучения весьма сложные формы и таким образом решать многие весьма сложные задачи.

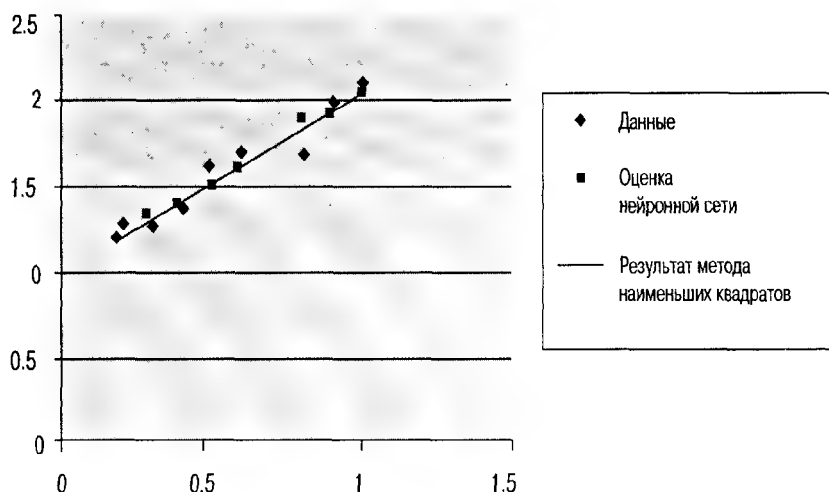


Рис. 1.16. Данные и соответствующая прямая, найденная с помощью нейронной сети, схема которой показана на рис. 1.15. Найденная нейронной сетью прямая мало отличается от прямой, получаемой при использовании метода наименьших квадратов

1.5. Резюме

- Нейронная сеть представляет собой совокупность простых обрабатывающих элементов, посылающих сигналы один другому по взвешенным связям.
- Типы связей, допустимых между элементами в сети, зависят от конкретной модели сети.
- Для каждого элемента сети имеется правило суммирования поступающих сигналов и правило вычисления выходного сигнала, по-

сылаемого затем другим элементам сети. Правило вычисления выходного сигнала называется функцией активности.

- Нейронная сеть может обучаться выполнению определенной задачи и обычно в привычном смысле не программируется. Обучение заключается в изменении значений весовых коэффициентов. В ходе обучения величина, на которую должен измениться весовой коэффициент, вычисляется с помощью соответствующего правила.

1.6. Дополнительная литература

Эта книга создавалась как введение в область нейронных сетей, ставящее своей целью подготовить читателя к пониманию более сложных публикаций. В дальнейшем в книге вы встретите множество ссылок на другие печатные издания. Некоторые из рекомендованных публикаций освещают очень узкие аспекты нейронных сетей. Среди прочих ссылок, читателю будут встречаться и ссылки на следующие наиболее интересные книги, которые настоятельно нами рекомендуются для более глубокого понимания предмета.

В книге Фосетта [Fausett, 1994] более подробно освещены темы, рассмотренные в первых шести главах нашей книги. Большинство посвященных нейронным сетям текстов использует много математических выкладок. Хотя книга Фосетта и является математической, рассматриваемые в ней вопросы освещаются довольно подробно, а объяснения сопровождаются хорошими примерами.

Книга Хейкина [Haykin, 1994] не для тех, кто боится математики. В ней с достаточной подробностью рассматриваются различные типы архитектуры сетей. Большинству студентов книга покажется слишком сложной, если использовать ее для знакомства с предметом, но она может быть превосходным справочным пособием, если вы пожелаете углубить свои знания в области нейронных сетей.

Книге Румелхарта и др. [Rumelhart *et al.*, 1986b] мы в значительной степени обязаны за возрождение интереса к нейронным сетям, последовавшим за довольно спокойным периодом исследований 70-х годов. Эта книга часто цитируется в изданиях, посвященных нейронным сетям. Хотя книга, вероятно, отчасти и устарела, в ней можно найти еще немало полезных идей, и ее стоит почитать хотя бы из-за той роли, которую она сыграла в истории развития нейронных сетей. В целом книга интересна, а автор книги, которую вы держите в руках, когда-то реализовал на языке С свою первую использующую алгоритм обратного распространения сеть, имея в качестве единственного учебника именно эту книгу (*и та программа действительно работала!*).

1.7. Упражнения

1. Для данных, представленных ниже, начертите на глаз несколько прямых, которые могут соответствовать этим данным. Запишите уравнения этих прямых, измерив соответствующие наклоны и координаты точек пересечения прямых с осью y . Для каждой прямой вычислите среднеквадратическую ошибку при условии, что для вводимых значений x вывод задается формулой

$$\text{вывод} = mx + c.$$

x	Требуемый вывод
0.30	1.60
0.35	1.40
0.40	1.40
0.50	1.60
0.60	1.70
0.80	2.00
0.95	1.70
1.10	2.10

2. Для данных из упражнения 1 найдите прямую, получаемую в результате применения метода наименьших квадратов.
3. Для данных упражнения 1 и заданных начальных весовых коэффициентов

$$\text{вывод} = 0.5x + 0.5$$

вычислите новую прямую после одного прохода через данные, используя правило обучения Видроу–Хоффа (дельта-правило) с нормой обучения, равной 0.3. (Замечание: после рассмотрения каждого учебного образца получается новая прямая.)

4. Наша подбирающая прямые линии нейронная сеть из раздела 1.4 имела один входной элемент, а целью было нахождение весовых коэффициентов, при которых по заданному x можно было бы оценить y . Как будет показано в главе 2, во многих задачах требуется знать, с какой стороны от прямой (т.е. выше или ниже ее) будет располагаться

точка данных. Представьте полученную в результате решения упражнения 2 прямую в следующем виде:

$$\text{ввод}(x, y) = mx - y + c.$$

Вычислите ввод для всех точек данных, определенных в формулировке упражнения 1, и для каждой точки найдите выходное значение, используя двоичную пороговую функцию. Как вы думаете, можно ли моделировать с помощью нейронной сети вычисление значений *ввода*?

Глава 2

Классификация образцов

Задача. Описание проблемы классификации и нейронных сетей, способных выполнять классификацию образцов.

Цели. Вы должны понять:

что такое проблема классификации и что такое управляемое обучение;

на основе чего выполняется классификация в сетях с прямой связью;

принципы построения сетей с прямой связью, в которых используется алгоритм обратного распространения ошибок, чтобы иметь возможность реализовать такую нейронную сеть на языке программирования, который вы предпочтете; разницу между линейной и нелинейной проблемами.

Требования. Знание основ линейной алгебры на уровне, представленном в приложении А. Для понимания некоторых разделов требуется знание основ дифференцирования, но это не слишком важно с точки зрения указанных выше целей. Знакомство с материалом главы 1.

2.1. Приложения

Многие приложения можно интерпретировать, как проблемы классификации. Например, если вы работаете администратором в банке и вам приходится решать, следует выдать кому-то кредит или нет, вы можете классифицировать всех потенциальных клиентов по уровню риска: низкий, средний или высокий риск. При оптическом распознавании символов сканируемые символы тоже ассоциируются с соответствующими им классами. Имеется немало вариантов изображения буквы “Н” даже для одного конкретного шрифта — символ может оказаться, например, смазанным, — но все эти изображения должны принадлежать классу “Н”. Распознавание слов в звуковой записи, воспроизводимой динамиком магнитофона, даст нам еще один из множества примеров классификации.

Когда мы знаем к какому классу относится каждый из учебных примеров, можно использовать управляемое обучение. Задачей для сети является ее обучение тому, как сопоставить предъявляемый сети образец с контрольным целевым образцом, представляющим нужный класс. Например, сети можно предъявить изображение буквы “Н” и обучить сеть тому, что при этом соответствующий “Н” выходной элемент должен быть включен, а выходные элементы, соответствующие другим буквам, — выключены. В этом случае входной образец может быть набором значений, характеризующих пиксели изображения в оттенках серого, а целевой выходной образец — вектором, значения всех координат которого должны быть равными 0, за исключением координаты, соответствующей выходному элементу, представляющему “Н” (значение этой координаты должно быть равным 1).

Основная часть материала этой главы посвящена описанию наиболее часто используемого сегодня типа нейронных сетей, а именно сетям с обратным распространением ошибок. Такие сети используются при решении целого ряда задач, начиная от распознавания объектов по сигналам гидролокатора до моделирования дислексии — проблемы, возникающей у некоторых людей при распознавании слов.

2.2. Основные идеи

В этом разделе рассматриваются основные понятия, необходимые для описания модели сети, использующей алгоритм обратного распространения ошибок, о котором пойдет речь в разделе 2.4.

2.2.1. Функция выбора решения

Чтобы описать, каким образом работает нейронная сеть, рассмотрим тривиальную задачу и используем для решения этой задачи самый простой тип сети.

Иллюстрация такой задачи классификации представлена на рис. 2.1. Задача состоит в выработке правил классификации самолетов для бомбардировщиков и истребителей в зависимости от их максимальной скорости и максимального взлетного веса. Такие правила могут быть заданы формально:

ЕСЛИ вес > 0.80 И скорость < 0.55 , ТО бомбардировщик,

ЕСЛИ вес < 0.90 И скорость > 0.25 , ТО истребитель.

Эти правила используют дискретные граничные значения, разделяющие пространство всех значений на прямоугольные области. Разделение, порожденное этими правилами, вполне успешно классифицирует самолеты, представленные на нашей диаграмме, но оказывается не слишком гибким, если по этим правилам придется классифицировать новый само-

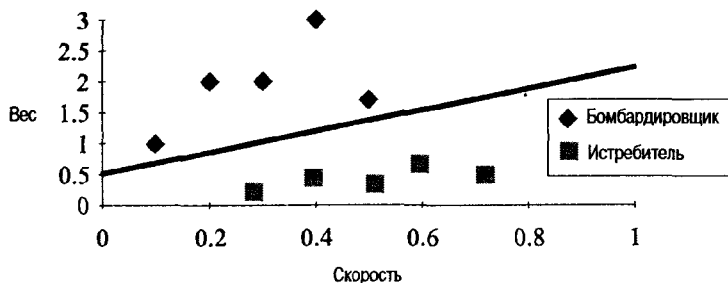


Рис. 2.1. Разделение абстрактных данных на два класса

лет. Кроме того, эти правила в указанном виде ничего не сообщают о том, насколько точной будет классификация нового самолета.

Альтернативный подход к использованию правил заключается в выводе функции классификации путем построения прямой, разделяющей два класса. Для нового самолета нам нужно просто указать точку на плоскости, соответствующую известным значениям максимальной скорости и максимального взлетного веса и посмотреть, по какую сторону от прямой будет расположена эта точка. В данном случае для небольшого количества самолетов рассмотрены два признака, скорость и вес, поэтому можно представить данные в виде изображения на плоскости. Однако, если придется иметь дело с сотнями самолетов и значительно большим числом признаков (т.е. в случае многомерной задачи), задачу классификации в виде простой картинки представить будет невозможно.

Выход заключается в использовании функции выбора решения. Уравнение прямой, разделяющей два типа самолетов, записывается в следующем виде:

$$x_2 = 1.5x_1 + 0.5,$$

где x_1 представляет скорость, а x_2 — вес. Это уравнение можно использовать для создания функции выбора решения:

$$f(x_1, x_2) = -x_2 + 1.5x_1 + 0.5,$$

$$d = \begin{cases} \text{истребитель.} & \text{если } f(x_1, x_2) \geq 0, \\ \text{бомбардировщик,} & \text{если } f(x_1, x_2) < 0. \end{cases}$$

Например, истребитель, представленный точкой (0.4, 0.5), даст

$$f(0.4, 0.5) = -0.5 + 1.5 \times 0.4 + 0.5 = 0.6,$$

и функция выбора решения правильно классифицирует эту точку, как истребитель.

Предлагаемую функцию выбора решения можно моделировать с помощью нейронной сети и даже реализовать в виде аппаратных средств. На рис. 2.2 показана сетевая модель для данной функции выбора решения. Сетевой ввод для центрального элемента находится путем умножения переменных ввода, x_1 и x_2 , на соответствующие их взвешенным связям коэффициенты с последующим суммированием результатов. Указанное на схеме значение смещения тоже добавляется в сумму, в результате чего получается значение комбинированного ввода для данного элемента. Как было показано в главе 1, вся сумма имеет следующий вид:

$$net_j = w_0 + \sum_{i=1}^n x_i w_{ij} ,$$

где net_j представляет значение комбинированного ввода, w_0 — смещение, связываемое с элементом, значение активности которого считается всегда равным 1, x_i — значение активности i -го элемента, а w_{ij} — вес связи, ведущей от элемента с номером i к элементу с номером j . Для элемента, показанного на рис. 2.2, выходное значение вычисляется согласно критерию пороговой функции:

$$\text{выход} = \begin{cases} 1, & \text{если комбинированный ввод} \geq 0, \\ 0, & \text{если комбинированный ввод} < 0. \end{cases}$$

Выходное значение 1 должно указывать на то, что самолет является истребителем, а значение 0 должно соответствовать бомбардировщику.

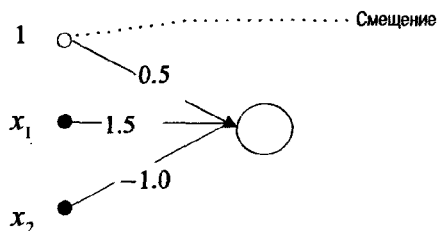


Рис. 2.2. Простая нейронная сеть

Ввод для сети на рис. 2.2 пропускается через пороговую функцию, в результате чего порождается выходное значение, равное 0 или 1. Вместо указанной пороговой функции можно было бы использовать любую другую функцию, дающую выходные значения в диапазоне от 0 до 1. Так, если выходное значение оказывается равным 0.9, мы можем быть практически уверены, что самолет является истребителем, а вот значение

0.5 не дает нам уверенности относительно его классификации. На самом деле систему можно использовать не только для классификации, а и для оценки степени точности такой классификации.

Пример 2.1

- (а) Вычислите комбинированный сетевой ввод для элемента на рис. 2.2 и соответствующее выходное значение при использовании пороговой функции и входного вектора $\{0.7 \ 2.5\}$
- (б) Вычислите выходное значение, используя в качестве функции активности сигмоидальную функцию. Входной вектор остается таким же, как и в п. (а).
- (в) Вычислите комбинированный ввод для сети с архитектурой, показанной на рис. 2.2, но с набором весовых значений $\{-0.2 \ 0.03 \ 1.2\}$ и таким же входным вектором, как и в п. (а).

Решение 2.1

- (а) Порядок элементов во входном векторе говорит о том, что $x_1 = 0.7$ и $x_2 = 0.5$. Таким образом, комбинированный ввод оказывается равным

$$0.5 + (0.7 \times 1.5) + (2.5 \times -1) = -0.95.$$

Комбинированный ввод оказывается отрицательным, поэтому вывод равен 0.

- (б) Активность в случае сигмоидальной функции вычисляется по следующей формуле:

$$f(\text{net}_j) = \frac{1}{1 + \exp(-\text{net}_j)}.$$

Комбинированный ввод равен -0.95 , и подстановка этого значения в сигмоидальную функцию дает выходное значение 0.28.

- (в) В этой книге в весовых матрицах соответствующие строкам индексы указывают на элементы, от которых исходят связи, а индексы, соответствующие столбцам, указывают на элементы, к которым эти связи направлены. Порядок размещения весовых значений в матрице весов означает, что смещение равно -0.2 , а для элемента, от которого исходит смещение, значение активности должно быть равным 1. Если входной вектор обозначить x , а вектор весов — w , то комбинированный ввод элемента можно выразить в виде

$$\text{net}_j = xw$$

при условии, что входной вектор включает и значение активности элемента смещения. Добавляя ко входному вектору значение активности элемента смещения, для комбинированного ввода в нашем случае получаем

$$\begin{aligned} net &= [1 \quad 0.7 \quad 2.5] \begin{bmatrix} -0.2 \\ 0.03 \\ 1.2 \end{bmatrix} \\ &= (1 \times -0.2) + (0.7 \times 0.03) + (2.5 \times 1.2) \\ &= 2.82. \end{aligned}$$

Пример 2.2

Найти весовые коэффициенты для модели нейронной сети, подобной показанной на рис. 2.2 и представляющей следующее уравнение:

$$2x_2 = 4x_1 + 8.$$

Решение 2.2

Для любой точки, лежащей на указанной прямой, весовые коэффициенты можно определить из уравнения

$$w_0 + x_1 w_1 + x_2 w_2 = 0.$$

Отсюда получаем

$$x_2 = -x_1 \frac{w_1}{w_2} - \frac{w_0}{w_2}.$$

Сравнивая члены полученного равенства с коэффициентами, указанными в условии примера, имеем

$$\frac{w_1}{w_2} = -\frac{4}{2}, \quad \frac{w_0}{w_2} = \frac{8}{2}.$$

Таким образом, $w_0 = -8$, а $w_2 = 2$.

2.2.2. Корректировка весов

Из рис. 2.1 должно быть ясно, что имеется целое множество прямых, которые могут быть выбраны в качестве границы, разделяющей данные, поэтому имеется целое множество весовых значений, дающих подходящее решение. Если требуемый выход i -го элемента обозначить t_j , а наблюдаемый на самом деле — o_j , то ошибка E_p для образца p может быть определена по формуле

$$E_p = \frac{1}{2} \sum_j (t_j - o_j)^2, \quad (2.1)$$

а полная ошибка будет равна $E = \sum E_p$. Множитель 1/2 включен здесь в формулу с целью упрощения выкладок в разделе 2.2.3.

Активность любого элемента зависит от комбинированного ввода этого элемента, а значит, от весовых значений, влияющих на этот элемент. Представьте себе элемент, подобный показанному на рис. 2.2, но без смещения. Такой элемент может моделировать любую прямую, проходящую через начало координат. В случае линейного элемента и одного образца равенство (2.1) можно записать в виде

$$E = \frac{1}{2} (t - net)^2,$$

так как для линейного элемента вывод оказывается равным вводу. Разворачивание правой части равенства дает

$$\begin{aligned} E &= \frac{1}{2} [t^2 - 2t \, net + net^2] \\ &= \frac{1}{2} [t^2 - 2t(x_1 w_1 + x_2 w_2) + x_1^2 w_1^2 + 2x_1 w_1 x_2 w_2 + x_2^2 w_2^2], \end{aligned} \quad (2.2)$$

где $net = x_1 w_1 + x_2 w_2$. Дифференцируя равенство (2.2) по w_1 , получаем

$$\frac{\partial E}{\partial w_1} = (-t + x_1 w_1 + x_2 w_2) x_1. \quad (2.3)$$

Равенство (2.2) говорит о том, что зависимость квадрата ошибки от w_1 является параболической, как показано на рис. 2.3, а если приравнять к нулю правую часть равенства в (2.3) и решить полученное таким образом уравнение, можно найти точку минимума соответствующей кривой.

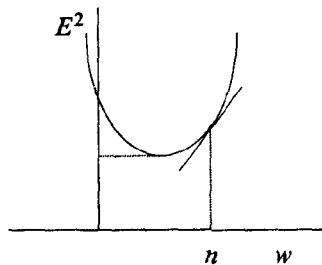


Рис. 2.3. Прямая линия представляет производную ошибки в зависимости от веса в момент времени n

Вспомните главу 1: перед началом обучения весовые коэффициенты устанавливаются равными некоторым случайным значениям. При этом точка, представляющая начальное состояние сети, может оказаться в любом месте на поверхности ошибок, но очень маловероятно, чтобы она оказалась в точке минимума этой поверхности. В процессе обучения сеть должна корректировать весовые коэффициенты так, чтобы максимально уменьшить значение общей ошибки. Другими словами, весовые коэффициенты должны корректироваться в том направлении, в котором спуск вниз по поверхности ошибок происходит быстрее всего. На рис. 2.3 эта идея иллюстрируется для одного веса, где n обозначает время или итерацию. Для случая двух весовых коэффициентов получается чашевидная поверхность ошибок, подобная показанной на рис. 2.4.

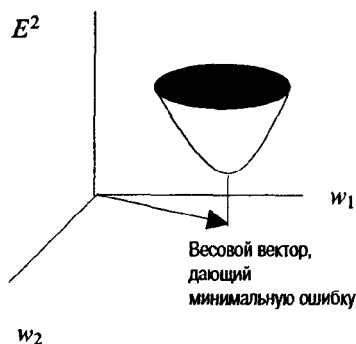


Рис. 2.4. Поверхность, представляющая величину ошибки для разных комбинаций весовых коэффициентов

2.2.3. Минимизация квадрата ошибки

Один из экспериментов, который многие из нас выполняли в школе, состоял в нанесении на координатную сетку точек, соответствующих величине изгиба металлической пластины в зависимости от приложенной к ней нагрузки, с последующим построением прямой, для которой сумма квадратов расстояний от полученных точек до прямой оказывалась минимальной. Тот же принцип можно использовать для корректировки весов, и одно из таких правил корректировки весов, называемое правилом Вилроу–Хоффа или дельта-правилом, уже упоминалось в главе 1. Это правило записывается в следующем виде:

$$\Delta w_{ij} = \eta \delta_j x_i, \quad \delta_j = (t_j - o_j), \quad (2.4)$$

где t_j обозначает требуемое значение для элемента j , o_j — его реальный вывод, x_i — сигнал, приходящий от элемента i , η — норма обучения (коэффициент, от которого зависит величина изменения веса), а Δw_{ij} — величина, на которую изменяется вес для связи, идущей от элемента i к элементу j .

Это правило очень просто получается в случае линейного элемента, когда вывод определяется следующей формулой:

$$o_j = \sum_i x_i w_{ij}.$$

Используя цепное правило, можно выразить производную поверхности ошибок в зависимости от веса в виде произведения, характеризующего изменение ошибки в зависимости от вывода элемента, и изменение вывода в зависимости от связанных с элементом весовых коэффициентов:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial w_{ij}}, \quad (2.5)$$

$$\frac{\partial E}{\partial o_j} = -\delta_j \quad (\text{это следует из (2.1) и определения } \delta_j \text{ в (2.4)}),$$

$$\frac{\partial o_j}{\partial w_{ij}} = x_i,$$

откуда, возвращаясь к (2.5), получаем

$$-\frac{\partial E}{\partial w_{ij}} = \delta_j x_i.$$

Принимая во внимание тот факт, что вес должен изменяться в направлении, противоположном направлению вектора градиента, и умножая на норму обучения, приходим к равенству (2.4).

Модификация этого метода корректировки весов будет рассмотрена в разделе 2.4, где она будет использоваться для обучения сетей с несколькими слоями элементов, но сначала мы должны обсудить преимущества таких сетей в сравнении с сетью, в которой имеется только один слой весовых значений (т.е. только слой входных элементов и слой выходных элементов).

2.3. Линейные и нелинейные проблемы

Сеть на рис. 2.2 имеет два входных элемента для двух признаков. Число признаков определяет размерность пространства, из которого выбираются все вводимые образцы: для двух признаков пространство ока-

зывается двумерным, для трех — трехмерным, а для n признаков пространство оказывается n -мерным. Простая модель сети, состоящей из трех входных и одного выходного элемента, будет моделировать плоскость, а модель с n входными элементами будет n -мерной гиперплоскостью. Для задачи классификации, например, выяснения типа самолетов, если прямая (для размерности два) или гиперплоскость (для размерности n) может разделить все образцы на соответствующие им классы, то проблема является *линейной*. Если же для решения проблемы разделения образцов на классы требуется несколько прямых или гиперплоскостей, то проблема называется *нелинейной*. Широко известным примером нелинейной проблемы является упоминавшаяся в главе 1 проблема моделирования отношения XOR. Отношение XOR при выводе дает 1 только тогда, когда в точности одно из вводимых значений равно 1, иначе вывод оказывается равным 0. Определение отношения XOR приведено в табл. 1.1. Проблема XOR, таким образом, является нелинейной, и для ее решения с помощью нейронной сети имеется две возможности: либо использовать сеть, которая будет строить две или больше прямых для разделения данных, либо изменить вид вводимых данных. Последняя возможность может превратить проблему в линейную, если к двум имеющимся вводимым признакам добавить третий и сделать пространство вводимых данных трехмерным (в результате два класса будут размещаться в двух противоположных вершинах куба). Мы на самом деле не будем считать последнюю возможность нейронным решением, поскольку добавление третьего признака означает, что мы вмешиваемся в процесс принятия решения, тем более, что для целого ряда сложных проблем такое вмешательство оказывается невозможным. Поэтому мы предпочитаем, чтобы сеть работала в условиях нелинейной проблемы. Таким образом, мы сосредоточимся на решении, использующем две прямых для разделения данных в их исходном двумерном виде. Такая сеть потребует два элемента, каждый из которых получит на входе по два значения, чтобы представить две разделяющие прямые, и третий элемент, объединяющий информацию об этих двух прямых. Проблема моделирования отношения XOR разделяющими границами иллюстрируется на рис. 2.5. На рис. 2.6 показана архитектура нейронной сети, которая будет моделировать две разделяющие границы.

Давайте выясним, что происходит с подаваемым на вход образцом в первом слое. Для элементов входного слоя будут использоваться индексы i , для элементов скрытого слоя — j , а для элементов выходного слоя — k . В сети на рис. 2.6 элементы разделены на три слоя. Элементы первого слоя являются элементами ввода данных в сеть. Не забывайте о том, что эти элементы отличаются от элементов последующих слоев тем, что они

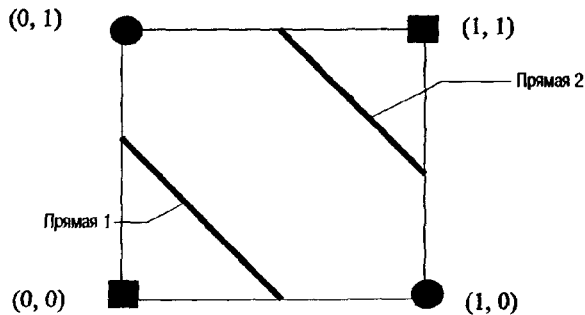


Рис. 2.5. Проблема XOR, решаемая с помощью двух разделяющих прямых

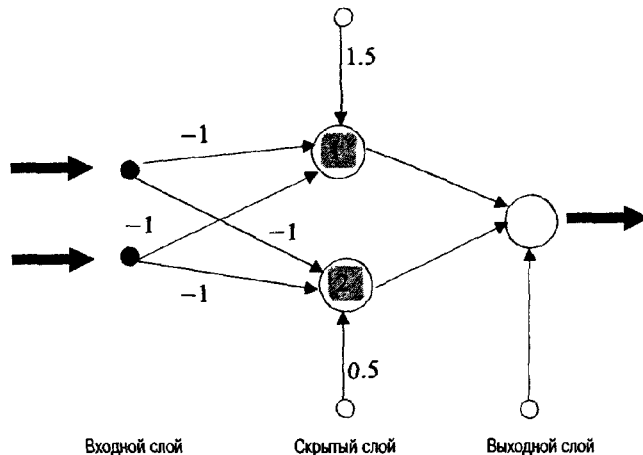


Рис. 2.6. Нейронная сеть для моделирования отношения XOR; второй слой весовых значений будет показан ниже

не имеют функции преобразования (т.е. вывод этих элементов оказывается равным вводу). Второй слой элементов называется *скрытым* слоем — скрытые элементы связаны только с другими элементами и не имеют непосредственных связей с внешней средой. Выходной слой предназначен для передачи ответа сети во внешнюю среду. В табл. 2.1 показаны комбинированный ввод и вывод (при использовании пороговой функции) для элементов скрытого слоя при ответе сети на вводимые данные проблемы XOR.

Таблица 2.1. Реакция скрытых элементов сети, показанной на рис. 2.6

x_1	x_2	Комбинированный ввод скрытого слоя		Вывод скрытого слоя	
		Элемент 1	Элемент 2	Элемент 1	Элемент 2
1	1	-0.5	-1.5	0	0
1	0	0.5	-0.5	1	0
0	1	0.53	-0.5	1	0
0	0	1.5	0.5	1	1

Вводимые образцы преобразуются под действием весовых коэффициентов первого слоя и скрытых элементов. Второй слой весов, соединяющих скрытый слой с выходным слоем, тоже будет моделировать прямую, поскольку у единственного выходного элемента также имеется два входных значения и значение смещения. Если выходной элемент правильно указывает класс, соответствующий каждому вводимому образцу, то данные ввода для второго слоя весов должны быть линейно отделимы. Мы можем проверить это с помощью нанесения на плоскость точек, соответствующих выводу скрытых элементов, как показано на рис. 2.7. На рис. 2.8 указаны весовые коэффициенты, для которых получается разделяющая прямая, показанная на рис. 2.7.

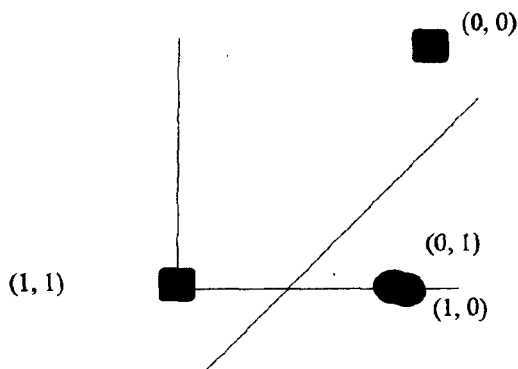


Рис. 2.7. В результате прохождения первого слоя весов исходная точка $(0, 1)$ переместилась в $(1, 0)$. Обратите внимание на то, что точки $(0, 0)$ и $(1, 1)$ поменялись местами

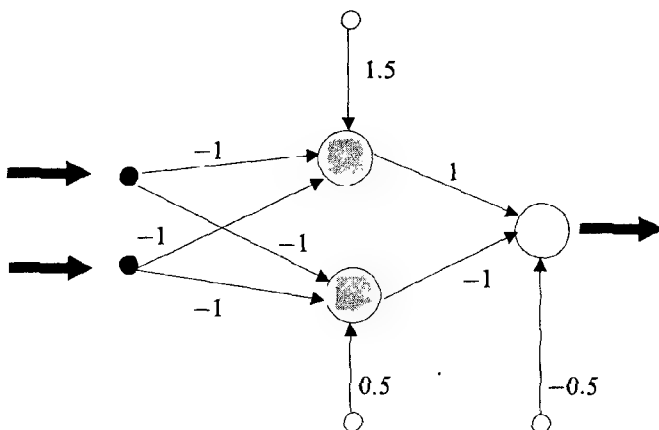


Рис. 2.8. Полная структура сети для решения проблемы XOR

Чтобы проверить правильность работы сети, необходимо проверить реакцию сети на каждый из вводимых образцов. Весовыми значениями первого слоя являются

$$\begin{bmatrix} 1.5 & 0.5 \\ -1.0 & -1.0 \\ -1.0 & 1.0 \end{bmatrix}$$

Если для весовых коэффициентов использовать обозначение w_{ij} , то вес w_{10} должен быть равным -1 , что соответствует весу связи, идущей от элемента 1 входного слоя к элементу 1 скрытого слоя, и располагаться в строке 2 и столбце 1 матрицы, поскольку индексация начинается с нуля. В табл. 2.2 для данных ввода XOR показан полный набор ответов сети, представленной на рис. 2.8.

Таблица 2.2. Ввод и вывод для сети типа 2-2-1

Входной вектор, p	Первый слой весов	Реакция скрытого слоя		Второй слой весов	Реакция выходного слоя	
		Комбинированный ввод	Вывод		Ввод	Вывод
$\begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1.5 & 0.5 \\ -1.0 & -1.0 \\ -1.0 & -1.0 \end{bmatrix}$	$\begin{bmatrix} -0.5 & -1.5 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \end{bmatrix}$	$\begin{bmatrix} -0.5 \\ 1.0 \\ -1.0 \end{bmatrix}$	-0.5	0

Окончание табл. 2.2

Входной вектор, p	Первый слой весов	Реакция скрытого слоя		Второй слой весов	Реакция выходного слоя	
		Комбинированный ввод	Вывод		Ввод	Вывод
$\begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 1.5 & 0.5 \\ -1.0 & -1.0 \\ -1.0 & -1.0 \end{bmatrix}$	$\begin{bmatrix} 0.5 & -0.5 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$	$\begin{bmatrix} -0.5 \\ 1.0 \\ -1.0 \end{bmatrix}$	0.5	1
$\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 1.5 & 0.5 \\ -1.0 & -1.0 \\ -1.0 & -1.0 \end{bmatrix}$	$\begin{bmatrix} 0.5 & -0.5 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$	$\begin{bmatrix} -0.5 \\ 1.0 \\ -1.0 \end{bmatrix}$	0.5	1
$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 1.5 & 0.5 \\ -1.0 & -1.0 \\ -1.0 & -1.0 \end{bmatrix}$	$\begin{bmatrix} 1.5 & 0.5 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$	$\begin{bmatrix} -0.5 \\ 1.0 \\ -1.0 \end{bmatrix}$	-0.5	0

Итак, мы вручную построили многослойную нейронную сеть, решающую классическую проблему XOR, но в действительности нас интересует создание механизма, который позволил бы сети научиться находить решение самостоятельно. Прежде чем обратиться к рассмотрению алгоритма обучения, дающего возможность сети научиться находить решения, необходимо подчеркнуть важность функции преобразования (функции активности), связываемой с каждым элементом. Примером нелинейной функции активности является пороговая функция. Примером линейной функции является тождественная функция, при которой вывод оказывается равным вводу. Ограничения сети с линейными элементами иллюстрируются на рис. 2.9. Показанная на рисунке сеть имеет два входных элемента, два скрытых элемента и три выходных элемента. Матрицы весовых значений включают весовые коэффициенты смещений и имеют вид

$$\begin{bmatrix} 1.0 & -2.0 \\ 2.0 & 0.5 \\ -3.0 & 1.0 \end{bmatrix}$$

для первого слоя и

$$\begin{bmatrix} 2.0 & 1.0 & 3.0 \\ -1.0 & 5.0 & 4.0 \\ -3.0 & 1.0 & 2.0 \end{bmatrix}$$

для второго.

Линейные и нелинейные проблемы

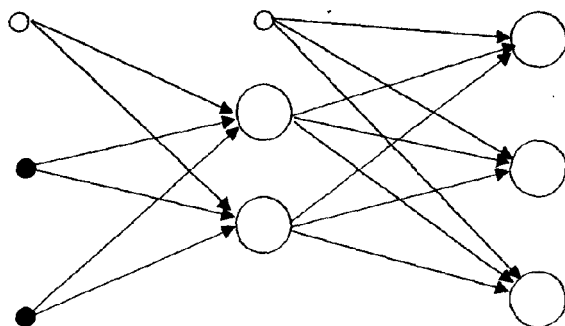


Рис. 2.9. Сеть типа 2-2-3

Для входного вектора $[1.0 \ 0.0 \ 1.0]$ ввод (а также и вывод, ввиду использования линейной функции преобразования) скрытых элементов задается следующим образом:

$$[1.0 \ 0.0 \ 1.0] \begin{bmatrix} 1.0 & -2.0 \\ 2.0 & 0.5 \\ -3.0 & 1.0 \end{bmatrix} = [-2.0 \ -1.0].$$

Ввод и вывод для выходного слоя получается в результате присоединения к скрытым элементам значения 1.0, равного значению активности смещения, и умножения на второй слой весов:

$$[1.0 \ -2.0 \ -1.0] \begin{bmatrix} 2.0 & 1.0 & 3.0 \\ -1.0 & 5.0 & 4.0 \\ -3.0 & 1.0 & 2.0 \end{bmatrix} = [7.0 \ -10.0 \ -7.0].$$

Рассмотрим теперь следующее произведение:

$$[1.0 \ 0.0 \ 1.0] \begin{bmatrix} 5.0 & 3.0 & 3.0 \\ -3.5 & 10.5 & 9.0 \\ 2.0 & -13.0 & -10.0 \end{bmatrix} = [7.0 \ -10.0 \ -7.0].$$

Оно дает тот же результат, что и предыдущее, являющееся результатом двух матричных умножений. Из ассоциативного закона умножения матриц следует, что в случае линейных функций активности можно найти один слой весов, дающий тот же результат, что и сеть с несколькими слоями. Другими словами, многослойная сеть с линейными функциями активности сможет решать только те проблемы, которые могут быть решены однослойной сетью (т.е. сетью, имеющей только входные и выход-

ные элементы). Таким образом из рассмотрения выпадают нелинейные проблемы. Итак, для многослойных сетей необходимо рассматривать нелинейные функции активности, а для алгоритма, к обсуждению которого мы собираемся приступить, такая функция должна быть непрерывной, дифференцируемой и монотонно возрастающей. Функцией, удовлетворяющей всем этим требованиям является логистическая функция (см. главу 1, рис. 1.10).

2.4. Обучение по алгоритму обратного распространения ошибок

На протяжении многих лет не знали правила, которое можно было бы использовать для корректировки весов многослойной сети в процессе управляемого обучения. В 70-х годах Вербос (Werbos) разработал подходящий алгоритм корректировки весов, но только Румелхарт и др. [Rumelhart *et al.*, 1986a] смогли дать новый толчок развитию нейронных сетей. Правило корректировки весов, о котором идет речь, называется *алгоритмом обратного распространения ошибок*. В дальнейшем в этом разделе мы будем рассматривать полносвязную сеть с прямой связью, т.е. сеть, в которой сигналы активности передаются только в направлении от входного слоя к выходному, а элементы одного слоя соединены со всеми элементами следующего слоя.

Алгоритм обратного распространения определяет два потока в сети: прямой поток от входного слоя к выходному и обратный поток — от выходного слоя к входному. Мы уже видели, что прямой поток продвигает входные векторы через сеть, в результате чего в выходном слое получаются выходные значения сети. Обратный поток подобен прямому, но он продвигает назад по сети значения ошибок, в результате чего определяются величины, в соответствии с которыми следует корректировать весовые коэффициенты в процессе обучения. В обратном потоке значения проходят по взвешенным связям в направлении, обратном направлению прямого потока. Например, в прямом потоке элемент скрытого слоя посылает сигналы каждому элементу выходного слоя, а в обратном потоке элемент скрытого слоя будет получать сигналы ошибок от каждого элемента выходного слоя. Наличие двойного потока в сети иллюстрирует рис. 2.10.

В процессе обучения каждый входной образец будет иметь соответствующий целевой выходной образец, который должен получаться для данного входного. Например, в случае проблемы XOR для входного образца [1.0 0.0] сеть должна давать на выходе 1.0, поэтому значение 1.0 будет целевым выходным значением для данного входного образца.

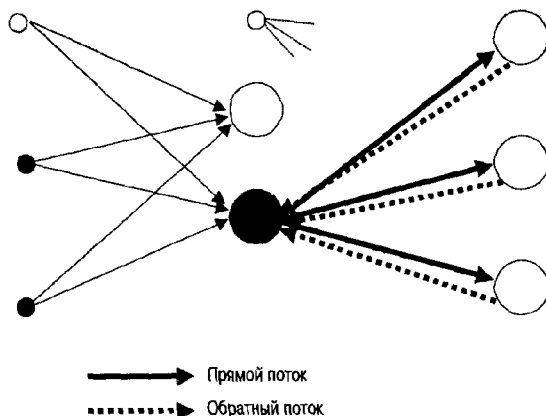


Рис. 2.10. Закрашенный скрытый элемент посылает сигнал активности каждому выходному элементу, поэтому в обратном потоке этот скрытый элемент получит сигналы ошибок от всех выходных элементов

В общем, целью обучения является нахождение такого набора весовых коэффициентов сети, который обеспечивает решение данной конкретной проблемы. Перед началом обучения весовые коэффициенты устанавливаются равными малым случайным значениям — например, значениям из диапазона от -0.3 до 0.3 . По причинам, которые уже обсуждались выше, будет использоваться нелинейная функция активности. Можно, например, рассмотреть сигмоидальную функцию. Сигмоид дает значения только между 0 и 1, а поскольку данная функция никогда не достигает значений 0 и 1, то, как правило, вместо 0 используют 0.1, а вместо 1.0 используют 0.9. Выбор конкретных значений 0.1 и 0.9 здесь не обязателен, поскольку обычно считается, что сеть научилась выполнять задачу, когда все выходные данные попадают в определенные допустимые рамки, содержащие целевые выходные значения. Например, если целевым выходным значением является значение 1.0, а допустимая погрешность равна 0.1, то любое выводимое значение между 0.9 и 1.0 будет находиться в допустимых рамках. Процедура обучения по правилу обратного распространения ошибок выглядит следующим образом.

для каждого входного вектора и связанного выходного вектора
 выполнять, пока not STOP
 STOP = TRUE
 для каждого входного вектора

выполнить прямой проход и найти реальный выход
 получить вектор ошибок путем сравнения реальных и целевых значений
 если реальный выход не попадает в допустимые рамки, установить STOP = FALSE
 выполнить обратный проход для вектора ошибок
 в результате обратного прохода определить величины изменения значений весов
 обновить значения весов

Коротко говоря, сети предъявляется образец и вычисляется вектор ошибок, в результате чего выясняется, насколько следует изменить значения весов; процесс повторяется для каждого образца. Полный цикл рассмотрения всех имеющихся образцов называется эпохой. Все образцы подаются на рассмотрение сети снова и снова, эпоха за эпохой, пока на протяжении одной эпохи все значения реального вывода для каждого образца не попадут в допустимые рамки.

2.4.1. Немного теории

Алгоритм обратного распространения ошибок опирается на обобщенные дельта-правила. В этом разделе мы используем представление, приведенное в [Rumelhart *et al.*, 1986a]. Для более глубокого анализа алгоритма обратитесь к [Rumelhart *et al.*, 1986a] или [Werbos, 1990].

Представим производную ошибки в виде

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}} \quad (2.6)$$

и определим δ_j с помощью формулы

$$\delta_j = -\frac{\partial E}{\partial net_j}. \quad (2.7)$$

Исходное дельта-правило из раздела 2.2.3 определяет δ_j как $\delta_j = -\partial E / \partial o_j$, но наше новое определение оказывается согласованным с исходным, так как исходное дельта-правило предназначается для линейных элементов, где выход оказывается равным вводу. Равенство (2.7) можно переписать в виде

$$\delta_j = -\frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j}.$$

Поскольку $E_p = \frac{1}{2} \sum_j (t_j - o_j)^2$, имеем

$$\frac{\partial E}{\partial o_j} = -(t_j - o_j).$$

Для функции активности f (обычно это логистическая функция) выходом является

$$o_j = f(\text{net}_j)$$

и поэтому для производной f' получаем:

$$\frac{\partial o_j}{\partial \text{net}_j} = f'(\text{net}_j).$$

Таким образом,

$$\delta_j = (t_j - o_j) f'(\text{net}_j).$$

Для нахождения комбинированного ввода используется обычное суммирование произведений:

$$\text{net}_j = \sum_{i=0} x_i w_{ij}.$$

Поэтому

$$\frac{\partial \text{net}_j}{\partial w_{ij}} = x_i.$$

Рассмотрев произведение полученных производных и возвращаясь к (2.6), получим

$$\frac{\partial E}{\partial w_{ij}} = -(t_j - o_j) f'(\text{net}_j) x_i.$$

С учетом того, что вес должен изменяться в направлении, противоположном тому, которое указывает производная поверхности ошибок, и с учетом нормы обучения η , изменение веса для элемента должно вычисляться по формуле

$$\Delta w_{ij} = \eta \delta_j x_i.$$

Указанная выше ошибка δ_j соответствует ошибке выходного элемента, но ошибка скрытого элемента не связана с целевым выходным значением непосредственно. Поэтому весовые значения скрытого элемента следует скорректировать пропорционально его “вкладу” в величину ошибки следующего слоя (т.е. выходного слоя в случае сети с одним скрытым слоем). В сети с одним скрытым слоем при распространении сигналов ошибок в обратном направлении ошибка каждого выходного элемента вносит свой “вклад” в ошибку каждого элемента скрытого слоя. Этот “вклад” для элемента скрытого слоя зависит от величины ошибки выходного элемента и весового коэффициента связи, соединяющей элементы. Другими словами, выходной

элемент с большей ошибкой делает больший “вклад” в ошибку того элемента скрытого слоя, который связан с данным выходным элементом большим по величине весом. Для скрытого элемента ошибка вычисляется по формуле

$$\delta_j = f'(net_j) \sum_k \delta_k w_{kj},$$

где индекс k соответствует слою, посылающему ошибку обратно (для сети с одним скрытым слоем это будет выходной слой).

Подходящей функцией активности является логистическая функция

$$f(net_j) = \frac{1}{1 + \exp(-net_j)}.$$

Производная этой функции активности равна

$$\begin{aligned} f'(net_j) &= \frac{\exp(-net_j)}{(1 + \exp(-net_j))^2} \\ &= \frac{1}{1 + \exp(-net_j)} \left(1 - \frac{1}{1 + \exp(-net_j)} \right) \\ &= f(net_j) [1 - f(net_j)]. \end{aligned}$$

2.4.2. Алгоритм обратного распространения ошибок

На первой стадии происходит инициализация весов малыми случайными значениями — например, значениями из диапазона между -0.3 и $+0.3$. Обучение предполагается управляемым, поскольку с каждым входным образцом связывается целевой выходной образец. Обучение продолжается до тех пор, пока изменение усредненной квадратичной ошибки не окажется меньше некоторого допустимого значения при переходе от одной эпохи к следующей. Например, допустимое значение 0.01 означает, что усредненная квадратичная ошибка соседних эпох не должна отличаться более, чем на ± 0.01 . Если в процессе обучения наступает момент, когда ошибка в сети попадает в рамки допустимого изменения, говорят, что наблюдается *сходимость*. Другим критерием окончания обучения можно считать наступление момента, когда выход для каждого учебного образца оказывается в рамках допустимого отклонения от соответствующего целевого выходного образца.

Чтобы уменьшить вероятность того, что изменения весов приобретут осциллирующий характер, вводится инерционный член α , добавляемый в пропорции, соответствующей предыдущему изменению веса:

$$\Delta w_{ij}(n+1) = \eta \delta_j o_i + \alpha \Delta w_{ij}(n)$$

Таким образом, изменение веса на шаге $n+1$ оказывается зависящим от изменения веса на шаге n . Алгоритм обратного распространения в целом представлен на рис. 2.11.

Ниже будут рассмотрены самые простые примеры соответствующих вычислений, но если представленный алгоритм покажется вам слишком сложным и непонятным, то в разделе 2.7 вы сможете найти пример подробных вычислений для конкретной сети. Эта сеть будет иметь два скрытых слоя, и вы сможете проследить за выкладками, что должно помочь вам разобраться в соответствующих вычислениях.

Пример 2.3

Представьте полностью прямой и обратный проходы в сети с прямой связью, использующей алгоритм обратного распространения ошибок, для входного образца $[0.1 \ 0.9]$ и целевого выходного значения 0.9 в предположении, что сеть имеет архитектуру 2-2-1 (т.е. два входных, два скрытых один выходной элемент) с весовыми коэффициентами

$$\begin{bmatrix} 0.1 & 0.1 \\ -0.2 & -0.1 \\ 0.1 & 0.3 \end{bmatrix}$$

для первого слоя и

$$\begin{bmatrix} 0.2 \\ 0.2 \\ 0.3 \end{bmatrix}$$

для второго слоя.

Решение 2.3

Выход для входных элементов совпадает с вводимыми значениями. Первые строки обеих весовых матриц определяют элементы смещения соответствующего слоя, которые, как вы помните, связываются с элементами, значения активности которых равны 1. Элементы получают номера $\{0, 1, 2\}$ для входного слоя, $\{3, 4, 5\}$ — для скрытого слоя и $\{6\}$ — для выходного слоя. При этом элементы с номерами 0 и 3 оказываются элементами смещения для входного и скрытого слоев соответственно.

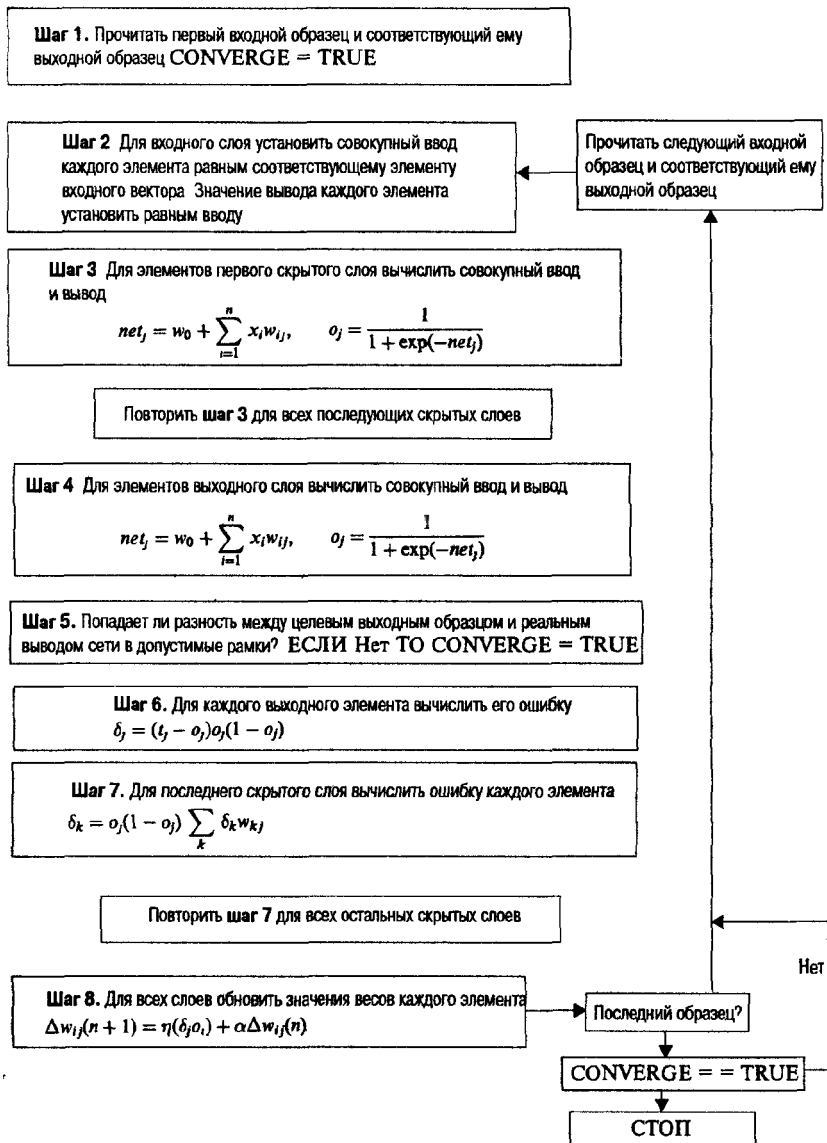


Рис. 2.11 Алгоритм обратного распространения ошибок. Индекс k соответствует предыдущему слою при обратном движении в сети. Используемые здесь обозначения подробнее описаны в приложении А

$$\begin{aligned} net_4 &= (1.0 \times 0.1) + (0.1 \times -0.2) + (0.9 \times 0.1) \\ &= 0.170, \end{aligned}$$

$$\begin{aligned} o_4 &= \frac{1}{1 + \exp(-0.17)} \\ &= 0.542, \end{aligned}$$

$$\begin{aligned} net_5 &= (1.0 \times 0.1) + (0.1 \times -0.1) + (0.9 \times 0.3) \\ &= 0.360, \end{aligned}$$

$$\begin{aligned} o_5 &= \frac{1}{1 + \exp(-0.36)} \\ &= 0.589. \end{aligned}$$

Точно так же при переходе от скрытого к выходному слою получим:

$$\begin{aligned} net_6 &= (1.0 \times 0.2) + (0.542 \times 0.2) + (0.589 \times 0.3) \\ &= 0.485, \\ o_6 &= 0.619. \end{aligned}$$

Ошибка выходного элемента равна

$$\begin{aligned} \delta_6 &= (0.9 - 0.619) \times 0.619 \times (1 - 0.619) \\ &= 0.066. \end{aligned}$$

Ошибки скрытых элементов равны

$$\begin{aligned} \delta_5 &= 0.589 \times (1.0 - 0.589) \times (0.066 \times 0.3) \\ &= 0.005, \\ \delta_4 &= 0.542 \times (1.0 - 0.542) \times (0.066 \times 0.2) \\ &= 0.003. \end{aligned}$$

Обратите внимание на то, что ошибки скрытых элементов используются при вычислении значений коррекции для первого слоя весов. Для элемента смещения ошибка не вычисляется, так как с элементом смещения скрытого слоя не связан никакой элемент из первого слоя. Норма обучения для этого примера была выбрана равной 0.25. Для первого образца нет необходимости добавлять инерционный член, поскольку для этого образца нет изменения предыдущего веса:

$$\begin{aligned} \Delta w_{5,6} &= 0.25 \times 0.066 \times 0.589 \\ &= 0.01. \end{aligned}$$

Новым весом является

$$0.3 + 0.01 = 0.31.$$

Далее получаем

$$\begin{aligned}\Delta w_{4,6} &= 0.25 \times 0.066 \times 0.542 \\ &= 0.009\end{aligned}$$

и

$$0.2 + 0.009 = 0.209$$

Наконец,

$$\begin{aligned}\Delta w_{3,6} &= 0.25 \times 0.066 \times 1.0 \\ &= 0.017\end{aligned}$$

и

$$0.2 + 0.017 = 0.217.$$

Читателю предлагается вычислить новые значения весов для первого слоя в качестве упражнения.

Пример 2.4

На рис. 2.12 показаны скрытый и выходной слои сети с прямой связью. Вычислите ошибку для скрытого элемента U при условии, что значение его активности для обрабатываемого сетью образца равно 0.64.

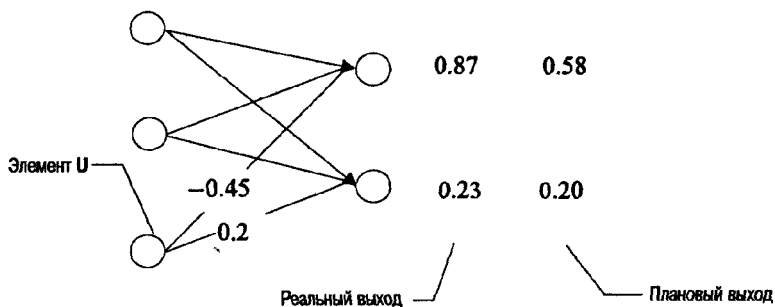


Рис. 2.12. Простая сеть. Входной слой не показан

Решение 2.4

Сначала вычисляются ошибки для выходных элементов:

$$\begin{aligned}\delta_{\text{выход}_1} &= (0.58 - 0.87) \times 0.87 \times (1 - 0.87) \\ &= -0.033,\end{aligned}$$

$$\begin{aligned}\delta_{\text{выход}_2} &= (0.20 - 0.23) \times 0.23 \times (1 - 0.23) \\ &= -0.005.\end{aligned}$$

Теперь ошибки распространяются обратно к элементу U:

$$\begin{aligned}\delta_U &= 0.64 \times (1 - 0.64) \times [(-0.033 \times -0.45) + (-0.005 \times 0.20)] \\ &= 0.003.\end{aligned}$$

2.4.3. Практические рекомендации

В этом разделе приводится ряд практических рекомендаций по использованию сетей, в которых применяется алгоритм обратного распространения ошибок, но читателю рекомендуется обратить внимание также на раздел 6.4, где данный вопрос обсуждается подробнее. Успех применения нейронной сети обычно требует проведения множества экспериментов. Есть целый ряд параметров, от правильного выбора которых зависит, насколько быстро будет найдено решение. Для сетей с прямой связью число скрытых элементов и слоев может быть различным. Несомненно, важным фактором является выбор учебных данных, и ему должно уделяться большое внимание, чтобы быть уверенным, что сеть будет обладать хорошими свойствами обобщения на данных, которые не предлагались ей в ходе обучения. Можно наблюдать успешную работу сети при использовании учебных данных и впоследствии обнаружить, что сеть не может правильно обрабатывать новые данные. Для каждого нового типа задач нет универсального способа нахождения решения. Иногда решить проблему не удастся, но неудача в попытке найти решение не означает, что задачу невозможно решить с помощью нейронного подхода. Хотя в нейронных сетях и используется метод "попыток и оценок результата", значение хорошего понимания решаемой проблемы и принципов работы нейронных сетей трудно переоценить.

Теория говорит о том, что сеть с одним скрытым слоем может представить любое непрерывное отображение вида

$$y = f(x).$$

Поскольку в форме такого отображения можно представить многие реальные проблемы, есть ли вообще необходимость использовать несколько скрытых слоев? Но иногда оказывается, что сеть с двумя скрытыми слоями обучить проще, а вариант сети с прямой связью, называемый автоассоциативной сетью, для решения задачи сжатия данных может иметь множество скрытых слоев (см. главу 4).

Алгоритм обучения в том виде, как он был представлен, использует пошаговое обновление набора весовых значений, что означает изменение весовых коэффициентов после рассмотрения каждого образца. Иногда, чтобы быстрее обучить сеть, оказывается более выгодным использовать пакетное обновление, когда ошибки для каждого элемента накапливаются в течение целой эпохи, и только после этого происходит корректиров-

ка весов этого элемента. Может оказаться выгодным использовать случайный порядок представления сетью учебных образцов от эпохи к эпохе — в таком случае потребуются поэтапное обновление весовых значений. Обычно лучший подход обнаруживается экспериментально.

Обобщение

Если сеть порождает правильный вывод для большинства вводимых образцов из набора тестовых данных, говорят, что сеть обладает хорошими свойствами *обобщения*. Предполагается, что набор тестовых данных в процессе обучения не использовался.

Если сеть хорошо обучена строить гладкое нелинейное отображение, ее возможности можно интерполировать на новые образцы, которые подобны, но не повторяют в точности те образцы, которые использовались в процессе обучения. Негладкое отображение возникает тогда, когда сеть перетренирована. В такой ситуации сеть скорее будет подобна памяти, находя подходящие выходные данные среди множества учебных образцов.

Качество обобщения сети зависит от набора учебных данных и архитектуры сети. Набор учебных данных должен быть характерным для решаемой проблемы, но важным также является и число скрытых элементов. Если скрытых элементов больше, чем требуется для обучения нужному отношению ввода-вывода, будет больше весовых коэффициентов, чем это необходимо, и, если процесс обучения продолжается слишком долго, в результате может наблюдаться слишком близкое отслеживание данных. Иногда в экспериментах с сетями различной архитектуры и продолжительности времени обучения в качестве временного тестового набора используют подмножество учебных данных. Главный набор тестовых данных при этом все равно резервируется для основного тестирования способности обобщения соответствующей сети.

Применение нейронных сетей представляет собой экспериментальный подход в прикладной науке. При этом для разработки сетей имеются некоторые общие рекомендации. В книге Хейкина [Haykin, 1994] используется результат из работы Баума и Хасслера [Baum, Haussler, 1989], дающий общие рекомендации относительно размеров набора учебных данных. Рекомендуется выполнение следующего неравенства:

$$N > \frac{W}{\epsilon},$$

где N обозначает число учебных образцов, W — число весовых коэффициентов в сети, а ϵ — долю ошибок, допустимую в ходе тестирования. Так, при допустимости 10% ошибок число учебных образцов должно быть в 10 раз больше числа имеющихся в сети весовых коэффициентов.

2.5. Использование сети с обратным распространением ошибок

В этой главе мы рассмотрим тип проблем, которые могут быть интерпретированы, как проблемы отображения вводимого образца в один из нескольких имеющихся классов. Для проблемы XOR число классов равно двум, поэтому для решения этой проблемы достаточно одного выходного элемента с двумя возможными состояниями (т.е. значение, меньшее 0,1, соответствует состоянию **ВЫКЛЮЧЕНО**, а значение, большее 0,9, соответствует состоянию **ВКЛЮЧЕНО**). Общим подходом является связывание с каждым классом своего выходного элемента. Если вводимый образец берется, скажем, из класса C_k , то выходной элемент U_k должен быть включен, в то время как все остальные выходные элементы — выключены.

2.5.1. Классификация чисел

Рассмотрим задачу классификации чисел от 0 до 9. В данном случае имеется 10 классов, поэтому в качестве целевого выходного вектора можно использовать вектор, состоящий из 10 элементов. Например, в качестве целевого вектора для числа 2 можно выбрать [0 0 1 0 0 0 0 0 0], означающий, что третий выходной элемент должен быть включен, а и все остальные — выключены.

В рассматриваемом здесь случае целевые выходные значения для чисел от 1 до 9 будут представляться векторами из 9 элементов, а числу 0 будет соответствовать вектор, все биты которого выключены. Например, если сеть решит, что ей представлено число 2, то на ее выходе второй выходной элемент окажется равным 1, а все остальные будут равны 0. Числа, которые должны быть выучены сетью, показаны на рис. 2.13. Каждое число представляется сеткой размером 9×7 , где серым пикселям соответствует 0, а черным — 1.

Для моделирования выбирается сеть типа 63-6-9: 9×7 входных элементов (по одному на каждый пиксель), шесть скрытых элементов и девять выходных элементов для целевых выходных векторов. Соответствующие пикселям значения отображаются во входной слой так, как показано на рис. 2.14.

Обучение сети продолжалось в течение 600 эпох, при этом норма обучения выбиралась равной 0,3, а коэффициент инерции был равен 0,7. Выходной элемент переводился во включенное состояние, если значение активности оказывалось больше 0,9, и в выключенное состояние, если значение активности оказывалось меньше 0,1.

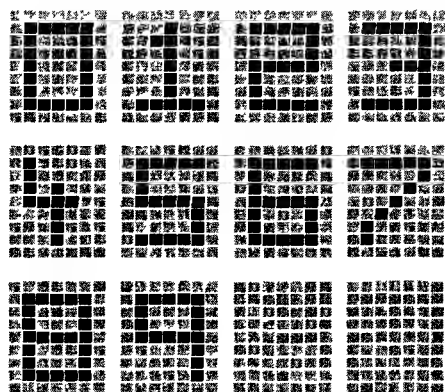


Рис. 2.13. Учебные данные

Обучение сети завершилось успешно. Затем сеть подверглась тестированию на данных, представленных на рис. 2.15. Каждое из чисел имело не менее одного пропущенного бита. Все тестовые числа были идентифицированы правильно, кроме числа 8, когда активность шестого выходного элемента оказалась равной значению 0.53, а восьмого — значению 0.41. Сеть не смогла решить, каким числом является восьмой образец — числом 6 или 8 — и поэтому не смогла отдать предпочтение ни одной из этих возможностей.

2.5.2. Классификация символов

Эта задача подобна предыдущей. Целью является обучение сети с прямой связью распознаванию символов A, B, C, D. Учебные символы были взяты из трех разных шрифтов. Для тестирования сети использовался четвертый тип шрифта. Все эти символы показаны на рис. 2.16. Они были построены по сетке размером 16×16 пикселей, поэтому для сети потребовалось 256 входных элементов. Число выходных элементов было равно четырем, по одному для каждого класса.

Подобная сеть будет успешно обучаться и правильно обобщать данные при разной архитектуре. Она легко обучается при наличии одного скрытого слоя из восьми элементов и правильно работает на указанном тестовом наборе символов. Такая сеть почти всегда успешно проходит обучение на указанных учебных данных. Однако число неудачных попыток тестирования оказывается примерно равным числу удачных, если использовать другие тестовые наборы. Неудачи обычно связаны с одним тестовым символом, который сеть классифицирует некорректно. Сеть имеет достаточно ресурсов, чтобы выполнить задание с восемью

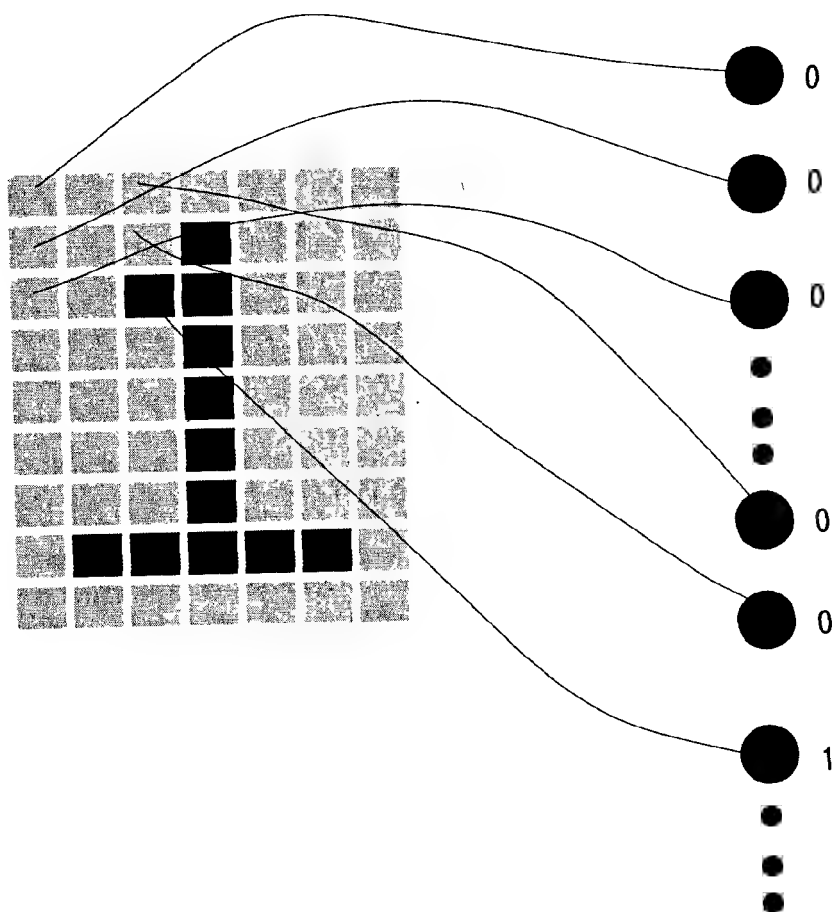


Рис. 2.14. Сетка значений отображается во входной слой с помощью ее представления в виде длинной строки битов, равных 0 или 1. Отображение значений битов начинается от верхнего левого угла сетки вниз до конца первого столбца и повторяется для каждого следующего столбца по порядку

или меньшим числом скрытых элементов. Как и ожидается, способность к обобщению ухудшается с ростом числа скрытых элементов, но иногда обобщение оказывается весьма хорошим для 100 или даже большего числа скрытых элементов.

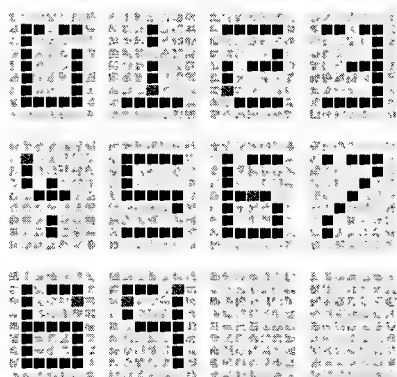


Рис. 2.15. Искаженные помехами тестовые данные

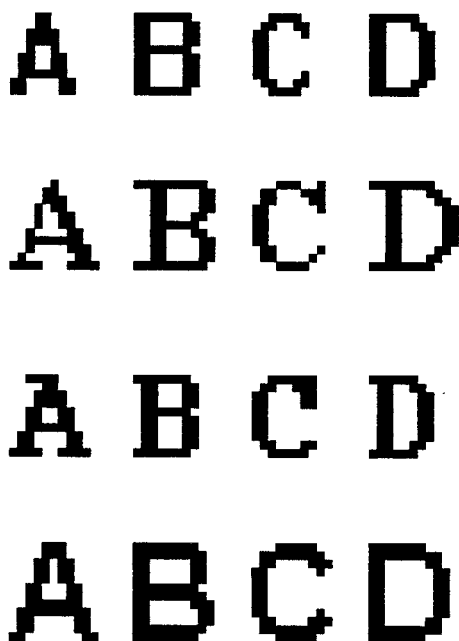


Рис. 2.16. Первые три ряда символов образуют учебный набор, а символы нижнего ряда – тестовый

2.5.3. Прогнозирование погоды

Чтобы заставить нейронную сеть предсказывать погоду с любой степенью надежности, вероятно, потребуются выполнить грандиозную исследовательскую работу. Это очень сложная проблема, и ученые потратили немало лет на разработку сложнейших компьютерных программ, помогающих строить прогнозы. Однако в качестве упражнения мы хотели бы обсудить то, каким образом можно было бы подойти к вопросу разработки системы предсказания погоды на завтра, основывающейся на информации о погоде сегодня. Такую систему можно назвать “наивным предсказанием”.

В табл. 2.3 представлены некоторые данные, полученные на одной из британских метеорологических станций по пяти признакам на протяжении 10 дней. Можно попытаться решить задачу прогнозирования, предоставляя сети информацию о погоде первого дня и считая целевыми выходными данными информацию о погоде второго дня. Следующим учебным образцом должны быть данные о погоде второго дня, а соответствующими целевыми выходными данными — данные третьего дня и т.д. Сеть должна иметь пять входных и пять выходных элементов. Можно начать с большого числа скрытых элементов (например, с 30), в дальнейшем пытаться уменьшить их число. Конечно, хотелось бы потребовать существенно больше данных, чем за 10-дневный период. Имея больше данных, можно было бы сформировать как учебный, так и тестовый наборы данных. Данные следовало бы предварительно обработать, чтобы все значения оказались в рамках допустимых значений активности элементов (для сигмоидальной функции это диапазон от 0.1 до 0.9). Данные в сыром виде, без предварительной обработки, оказываются неподходящими для использования. Такие данные не только не попадают в область допустимых значений функции активности, но наблюдается также очень большой разброс значений между признаками. Например, давление характеризуется значениями порядка 1000, тогда как характеристики дождя выражаются в значениях, меньших 10. Мы бы не хотели, чтобы данные, касающиеся давления, оказались в сети более значимыми, чем данные, касающиеся уровня осадков, поэтому и требуется предварительная обработка данных. Предварительная обработка данных обсуждается в главе 6.

Конечно, нельзя быть уверенным, что мы сможем разработать систему прогнозирования погоды с помощью указанного выше простого подхода, но в такой попытке и вреда не будет. По крайней мере, мы можем обнаружить, что неплохой краткосрочный прогноз (т.е. прогноз на завтра) сделать вполне возможно. На самом деле немало людей пытаются решить эту задачу. Знание особенностей проблемы прогнозирования по-

Таблица 2.3. Погодные данные

День	Температура (°C)		Осадки (мм)	Давление (мбар)	Солнце (часы)
	минимум	максимум			
01	-1	4.8	0.7	1011	3.8
02	-1	2.8	0	1024	5.4
03	-5.3	3.6	0	1032	4.8
04	-5	6.6	2.8	1026.5	0
05	3.5	4.7	3	1019.5	0
06	1.5	7.9	0	1018.5	5.2
07	-1.4	9.5	0.2	1034.5	0
08	0.8	10.4	1.4	1028.5	0.2
09	1.8	10.4	0.4	1028.5	0
10	4	12	2	1020.5	0

годы помогло бы нам формализовать проблему и выделить наиболее полезные данные. Например, вполне разумным кажется предположение о том, что лучший прогноз можно получить на основе непрерывного мониторинга изменения данных в течение всего дня, а не на суммарных данных об уровне осадков, максимуме температуры и т.д.

Задача построения прогноза погоды типична для нейронных сетей. В ней имеется информация, которую мы считаем характерной для рассматриваемой проблемы, но трудность состоит в том, что не ясно, как следует использовать данные и как выбрать подходящую архитектуру сети, с помощью которой должна эта проблема решаться. Здесь следует начинать с экспериментальной проверки идей, которые могли бы в данном случае оказаться работоспособными. Через эксперимент с данными приходит лучшее понимание проблемы и возникают новые идеи, открывающие новые возможности для ее решения.

2.6. Сети с радиальными базисными функциями

В этом разделе мы рассмотрим другой тип модели сети, которую тоже можно использовать для классификации образцов и которая называется сетью с радиальными базисными функциями.

Сеть с радиальными базисными функциями в наиболее простой форме представляет собой сеть с тремя слоями: обычным входным слоем, выполняющим распределение данных образца для первого слоя весов, скрытым

слоем и выходным слоем. Отображение от входного слоя к скрытому слою является нелинейным, а отображение скрытого слоя в выходной — линейно. Обычно (но не всегда) число скрытых элементов больше числа входных элементов. Идея здесь в том, что если нелинейную проблему разместить в пространстве высокой размерности некоторым нелинейным образом, то вероятность того, что она будет линейно отделимой, должна оказаться выше.

Рассмотрим отображение первого слоя. С каждым скрытым элементом связывается некоторая функция φ . Каждая из этих функций берет комбинированный ввод и порождает значение активности, подаваемое на выход. Совокупность значений активности всех скрытых элементов определяет вектор, на который отображается входной вектор

$$\varphi(\mathbf{x}) = [\varphi(x_1), \varphi(x_2), \dots, \varphi(x_M)],$$

где M обозначает число скрытых элементов, а $\mathbf{x}$ — входной вектор.

Связи элемента скрытого слоя определяют центр радиальной функции для данного скрытого элемента. Ввод для каждого элемента выбирается равным евклидовой норме

$$\begin{aligned} net_j &= \|\mathbf{x} - \mathbf{w}_j\| \\ &= \left[\sum_{i=1}^n (x_i - w_{ij})^2 \right]^{1/2}, \end{aligned}$$

где n обозначает число входных элементов.

Для скрытых элементов используют самые разные функции активности. Брумхед и Лауэ [Broomhead, Lowe, 1988] использовали функцию Гаусса $\varphi(r) = \exp[-r^2]$ или функцию $\varphi(r) = \sqrt{c^2 + r^2}$.

Пример 2.5

Сеть типа 2-2-1 с радиальными базисными функциями используется для решения проблемы XOR. Первый слой весов задан матрицей

$$\begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$$

Для каждого вводимого образца XOR вычислите значения активности для всех скрытых элементов, если функция активности имеет вид $\varphi(net) = \exp[-net^2]$, где net является евклидовой нормой.

Решение 2.5

Для образца (0, 1) и первого скрытого элемента получаем

$$\varphi_1 = \exp[-((0-1)^2 + (1-1)^2)] = \exp[-1] = 0.368$$

Для образца (0, 1) и второго скрытого элемента получаем

$$\varphi_2 = \exp[-((0-0)^2 + (1-0)^2)] = \exp[-1] = 0.368$$

Весь набор значений активности выглядит следующим образом

Ввод	φ_1	φ_2
(0 1)	0.368	0.368
(1 0)	0.368	0.368
(0 0)	0.135	1
(1 1)	1	0.135

Если нанести значения активности невидимых элементов на плоскость, будет видно, что теперь образцы оказываются линейно-отделимыми

Существует несколько методов определения весовых коэффициентов. Первый слой весов перед началом обучения может быть задан в результате случайного выбора центров функций каждого скрытого элемента из набора учебных данных. Первый слой весов обычно находится с использованием техники обучения без управления (см. главу 3). Как только найден первый слой весов, второй слой можно определить непосредственно, а если это не удастся, то с помощью линейного алгоритма управляемого обучения.

Одну и ту же проблему можно решить как с помощью сети с радиальными базисными функциями, так и с помощью многослойной сети с обратным распространением ошибок. Эти сети представляют собой вычислительные схемы довольно общего вида, применимые при решении очень широкого круга задач. Давайте рассмотрим пример использования сети с радиальными базисными функциями для решения задачи аппроксимации кривой. Кривая, которую нужно аппроксимировать, задается функцией

$$f(x) = 0.5x + 2x^2 - x^3,$$

а ее график показан на рис. 2.17.

Эта кривая будет аппроксимироваться с помощью взвешенных сумм функций Гаусса, имеющих вид

$$net_j = \exp\left[-\frac{1}{2\sigma^2}(c-x)^2\right],$$

где c задает центр функции. Пример кривой Гаусса, имеющей центр в нуле, показан на рис. 2.18. От значения константы σ зависит ширина колокола кривой.

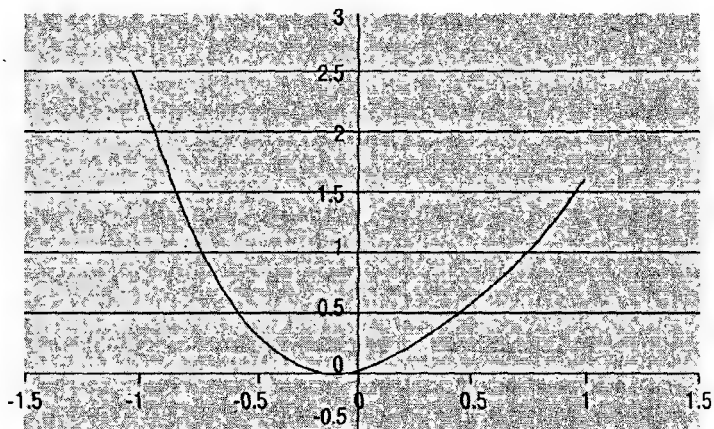


Рис. 2.17. График $f(x) = 0.5x + 2x^2 - x^3$

Эта кривая будет аппроксимироваться с помощью взвешенных сумм функций Гаусса, имеющих вид

$$\text{net}_j = \exp\left[-\frac{1}{2\sigma}(c-x)^2\right],$$

где c задаст центр функции. Пример кривой Гаусса, имеющей центр в нуле, показан на рис. 2.18. От значения константы σ зависит ширина колокола кривой.

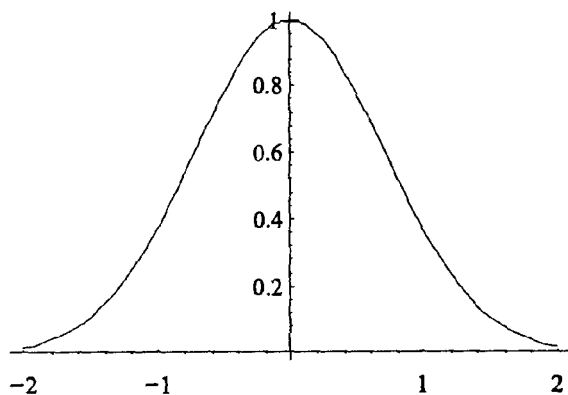


Рис. 2.18. Функция Гаусса

Число базисных функций выбирается произвольным образом. Были выбраны девять функций с центрами в точках $\{-0.8, -0.6, -0.4, -0.2, 0, 0.2, 0.4, 0.6, 0.8\}$. Ширина функций была выбрана равной 0.5. Кривая аппроксимируется взвешенной суммой базисных функций. Эти функции Гаусса представляют скрытый слой сети, поэтому скрытый слой состоит из девяти элементов. Первый слой весов, соответствующий связям, идущим от единственного входного элемента, представляет собой набор центров выбранных функций. Второй слой весов находится с помощью правила обучения Видроу–Хоффа, представленного в главе 1: для выхода o и целевого выхода t ошибка δ определяется формулой

$$\delta = t - o.$$

Суммарный сигнал, поступающий к выходному элементу, обозначен net . По закону обучения Видроу–Хоффа (дельта-правило) вносимая коррекция должна быть следующей:

$$\Delta w = \eta \delta net.$$

Архитектура сети показана на рис. 2.19. Норма обучения была выбрана равной 0.1. Чтобы найти второй слой весов, был сгенерирован 21 учебный образец путем пропускания некоторого числа (из диапазона между -1 и $+1$) через первый слой весов. Значения активности скрытого слоя затем давали вектор с девятью элементами для каждой учебной точки. Полученные в результате 21 вектор формировали набор учебных образцов для линейной сети, а соответствующие целевые выходные данные были значениями оригинальной кривой. Аппроксимация, построенная сетью, вместе с графиком оригинальной кривой показана на рис. 2.20.

На практике необходимо определить весовые значения сети и провести тестирование, в ходе которого проверяется качество работы сети. После этого можно скорректировать сначала первый слой весов, а затем — второй, чтобы попытаться улучшить качество. Процесс экспериментирования с параметрами сети продолжается до тех пор, пока качество работы сети не станет приемлемым.

2.7. Резюме

В этой главе мы говорили об управляемом обучении, при котором рассматривается отображение входного образца в целевой выходной образец. Для управляемого обучения необходимо иметь:

- данные, имеющие известную классификацию;
- данные в объеме, достаточном для представления всех аспектов решаемой проблемы;
- данные в объеме, достаточном для проведения тестирования.

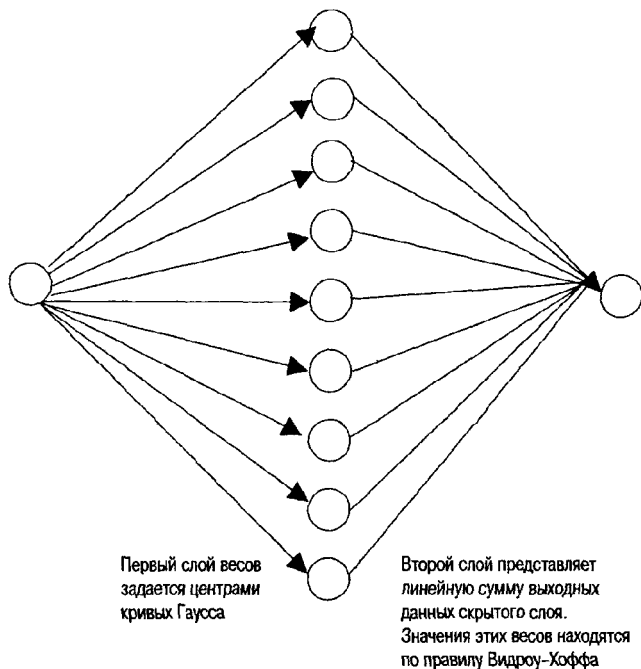


Рис. 2.19. Сеть с радиальными базисными функциями, обученная представлять кривую, как показано на рис. 2.17

Наборы тестовых и учебных данных должны быть разными (чтобы тестовые данные не использовались в ходе обучения). Знание решения задачи отражается в наборе весовых значений сети.

Очень часто используется алгоритм обратного распространения ошибок.

- В этом алгоритме обучение происходит путем корректировки весовых коэффициентов на основе применения обобщенного дельта-правила, когда осуществляется попытка минимизировать квадратичную ошибку, соответствующую отклонению реального вывода сети от требуемого вывода.
- В ходе обучения данные циклически обрабатываются снова и снова до тех пор, пока ошибка не станет достаточно малой для того, чтобы задача считалась решенной. Но даже если квадратичная ошибка оказывается малой по величине, важно проверить, чтобы все учебные образцы классифицировались правильно.

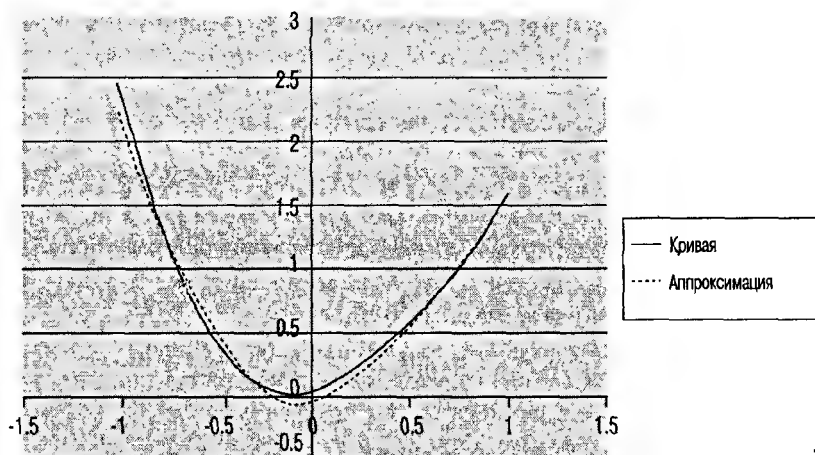


Рис. 2.20. Аппроксимация кривой с помощью сети с радиальными базисными функциями, представленной на рис. 2.19

- После завершения обучения значения весовых коэффициентов в сети фиксируются, и сеть может использоваться для классификации новых неизвестных ей образцов.
- Теоретически для моделирования многих проблем может быть достаточно одного скрытого слоя элементов, но на практике несколько скрытых слоев могут дать лучшее качество выполнения задачи.
- Сети с обратным распространением ошибок могут требовать длительного процесса обучения.
- Обобщение является мерой качества работы сети на данных, не представленных ей в ходе обучения. Число скрытых элементов и длительность обучения могут оказывать существенное влияние на способности обобщения сети.

Сеть с радиальными базисными функциями представляет собой другой тип сети, используемой для классификации (или моделирования функций общего вида).

- Для нее требуются данные в объеме, достаточном для представления всех аспектов решаемой проблемы.
- Для нее требуются данные в объеме, достаточном для проведения тестирования. Наборы тестовых и учебных данных должны быть разными (т.е. тестовые данные не должны использоваться в ходе обучения).

- Сеть решает нелинейную проблему путем размещения вводимых образцов в пространстве более высокой размерности некоторым нелинейным образом.
- Выбор функции активности для скрытых элементов должен быть сделан до начала обучения. Типичным является использование функции Гаусса.
- Центры функций Гаусса определяют первый слой весов. Они могут быть определены с помощью метода кластеризации (см. главу 3).
- Второй слой весов сети может быть определен с использованием линейного правила управляемого обучения.

Главная тема этой главы — использование сети с обратным распространением ошибок, поэтому давайте рассмотрим еще один пример, представив его в виде полной диаграммы прямого и обратного проходов в сети. Вводимыми данными будут (0 1, 0.9), норма обучения равна 0.8, а коэффициент инерции равен 0. Сеть имеет архитектуру 2-2-2-1, а весовые значения задаются матрицами

$$\begin{bmatrix} 2 & 2 \\ -2 & 3 \\ -2 & 3 \end{bmatrix}, \begin{bmatrix} 3 & -2 \\ -2 & 2 \\ -4 & 2 \end{bmatrix}, \begin{bmatrix} -2 \\ 3 \\ 1 \end{bmatrix},$$

где каждая матрица представляет свой слой весовых значений. На рис. 2.22 показан прямой поток данных в сети с указанными суммарным вводом и суммарным выводом каждого элемента. Обратный поток ошибок показывает взвешенные ошибки, получаемые элементами, и общую ошибку, вычисляемую по формуле, представленной на рис. 2.21. На рис. 2.23 представлены обновленные весовые значения.

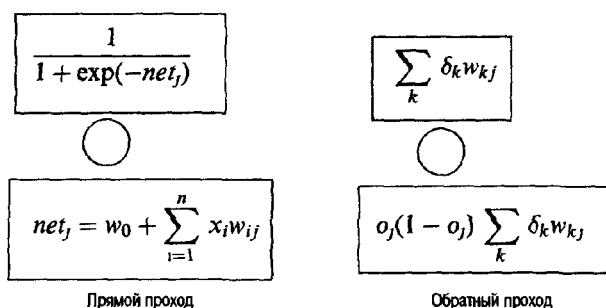


Рис. 2.21. Числа перед элементами и после них на рис. 2.22 соответствуют вычислениям по формулам, заключенным здесь в рамки

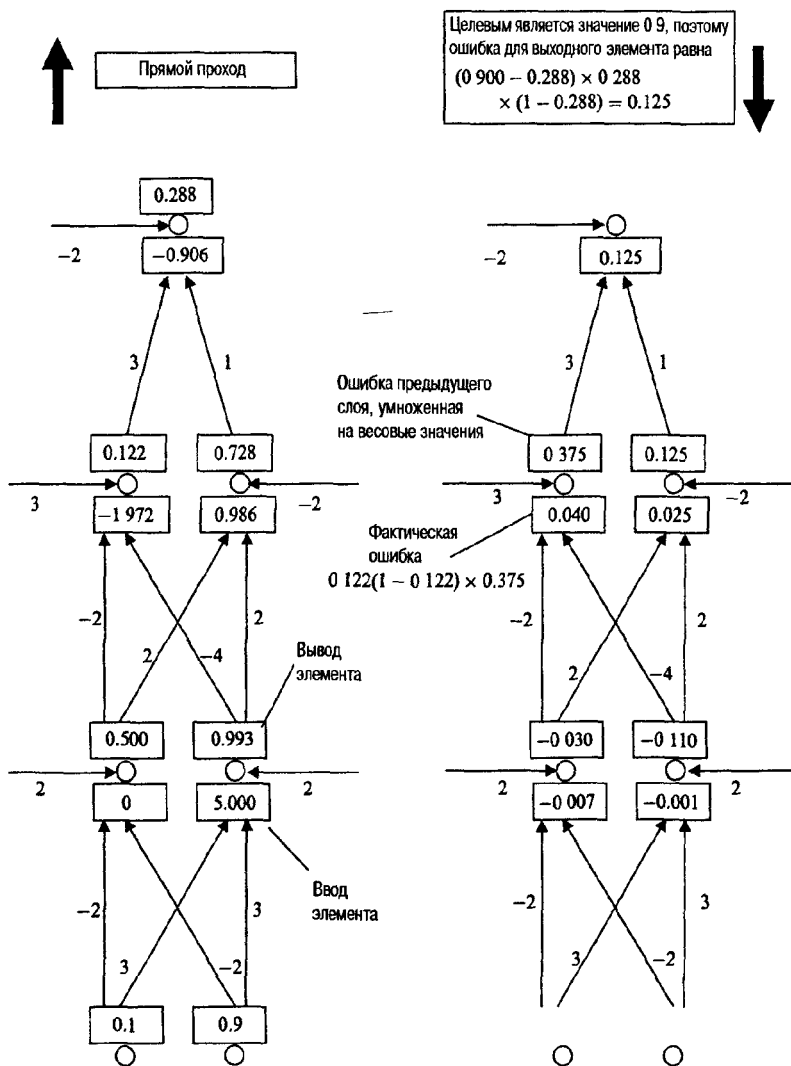


Рис. 2.22. Пример прямого и обратного проходов в сети типа 2-2-2-1 с прямой связью. Данные ввода, вывода и значения ошибок показаны в рамках (см. рис. 2.21)

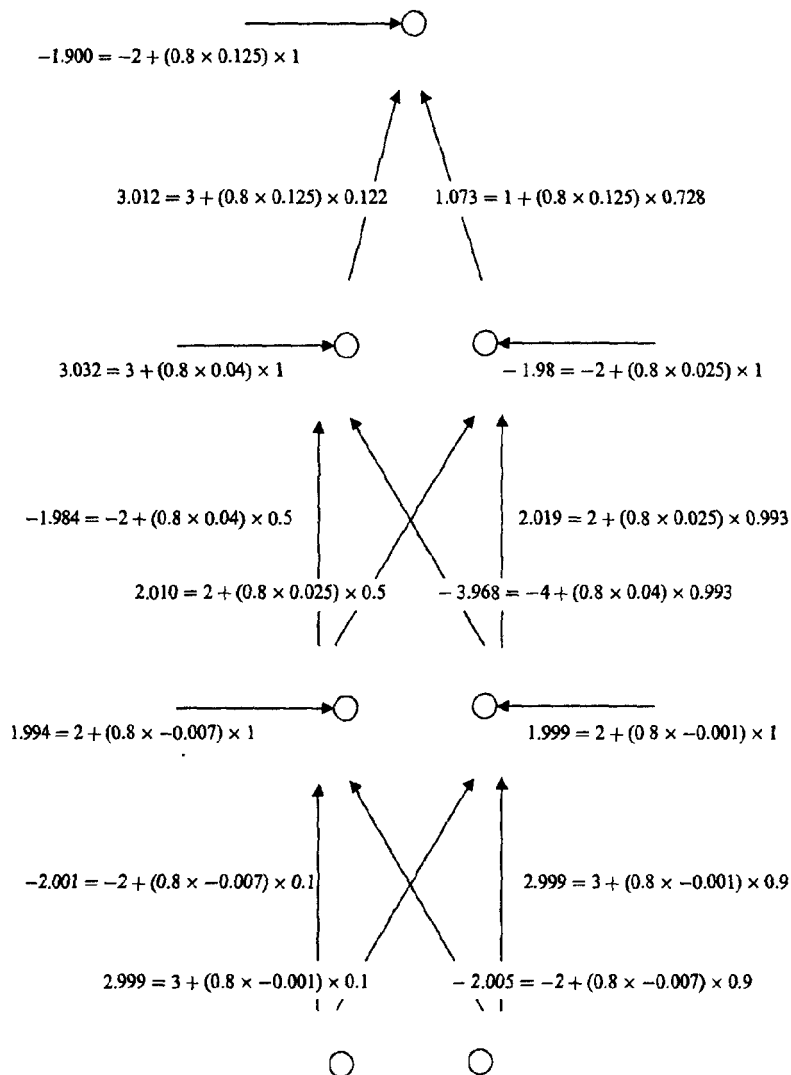


Рис. 2.23. Новые весовые значения, вычисленные по значениям ошибок, показанных на рис. 2.22

2.8. Дополнительная литература

Оригинальное представление алгоритма обратного распространения из [Rumelhart *et al.*, 1986a] до сих пор остается достойным изучения. Более поздняя статья [Werbos, 1990] также разъясняет теорию, на которую опирается данный алгоритм и его вариация для случая рекуррентных связей (см. главу 5). По поводу сетей с радиальными базисными функциями следует ознакомиться со статьёй [Broomhead, Lowe, 1988] и книгой [Haykin, 1994], которая является достаточно подробным введением в предмет.

Существует много вариантов алгоритма обратного распространения, ставящих целью ускорение обучения и помогающих сети избежать локальных минимумов, которые могут не позволить успешно завершить обучение. Некоторые из этих вопросов обсуждаются в главе 6, но любопытному читателю мы снова рекомендуем книгу [Haykin, 1994], а также [Masters, 1995], где предлагается практическое решение этих проблем с использованием программ на языке C++.

2.9. Упражнения

1. На рис. 2.24 показана сеть с обратным распространением ошибок во время обработки учебного вектора $[1 \ 0 \ 0.9 \ 0.9]$, для которого целевым выходным вектором является $[0.1 \ 0.9 \ 0.1]$. Пусть выходом элемента В является значение 0.6, а выходом элемента С — значение 0.8. Предположим, что функцией активности является сигмоид.

- (а) Вычислите фактический выходной вектор.
- (б) Вычислите значения ошибок для каждого выходного элемента.
- (в) Вычислите значения ошибок для каждого скрытого элемента.

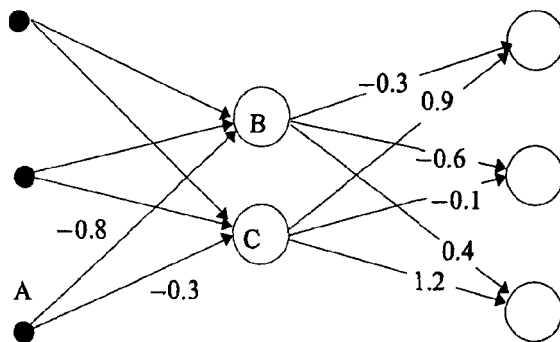


Рис. 2.24. Сеть типа 3-2-3, в которой нет элементов смещения

- (г) Вычислите изменения весовых значений для связей, идущих от элемента А. Норма обучения предполагается равной 0.25
2. Повторите вычисления упражнения 1 для целевого выходного вектора [0 1 0 9 0 9]
3. Предположим, что точки $\{(-1, 1), (-1, -1), (1, -1)\}$ принадлежат классу А, а точки $\{(-2, -2), (1, 1), (2, 2), (4, 1)\}$ — классу В
- (а) Докажите, что эти классы не являются линейно отделимыми.
- (б) Предположив, что выход элементов сети задается условием

$$\text{выход} = \begin{cases} 1, & \text{если комбинированный ввод} \geq 0, \\ 0, & \text{если комбинированный ввод} < 0, \end{cases}$$

покажите, что определенный ниже матрицей W_1 первый слой весовых значений в сети с тремя слоями преобразует проблему в линейную (первая строка матрицы W_1 определяет весовые коэффициенты смещения):

$$W_1 = \begin{bmatrix} 1 & -6 \\ -2 & -2 \\ -1 & -3 \end{bmatrix}.$$

- (в) Определите значения второго слоя весов так, чтобы сеть правильно классифицировала указанные выше образцы. Предположите, что сеть имеет один выходной элемент.
4. Точки $\{(4, -1), (8, -2), (1, 1), (3, 6)\}$ принадлежат классу А, а точки $\{(-8, 4), (-2, -3), (-1, -1), (2, -9)\}$ — классу В. Постройте минимальную сеть, правильно классифицирующую эти точки.
5. Покажите, что сеть с радиальными базисными функциями из примера 2.5 может решить проблему XOR, если значения первого слоя весов установить равными

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

6. Постройте второй слой весов для решения упражнения 5. Предположите наличие смещения для выходного элемента
7. Определите сеть с радиальными базисными функциями, решающую проблему XOR, в предположении, что функции активности скрытых элементов имеют вид $\varphi(r) \approx \sqrt{c^2 + r^2}$.
8. Повторите вычисления прямого и обратного проходов в сети, показанной на рис. 2.22, и найдите новые весовые значения для входного образца [0.9 0.9] и целевого выходного значения [0 1]

9. Постройте подходящую сеть с прямой связью, моделирующую операцию логического AND.
10. Для сетей с обратным распространением ошибок другой часто используемой функцией активности является двухполюсный сигмоид. Эта функция имеет область значений $(-1, 1)$ и определяется формулой

$$f(x) = \frac{2}{1 + \exp(-x)} - 1.$$

Производную этой функции можно выразить в виде

$$f'(x) = \frac{1}{2}[1 + f(x)][1 - f(x)].$$

Предложите правило коррекции ошибок для выходных и скрытых элементов.

11. Постройте график производной функции-сигмоида и объясните, как связаны относительная величина обновления веса и значение активности элемента.
12. На рис. 2.25 показаны два класса, A и B, образующие треугольные области. Третий класс определяется как пересечение классов A и B. Предложите архитектуру сети с обратным распространением ошибок, решающей такую задачу классификации. Объясните, почему вы выбрали именно такую архитектуру.

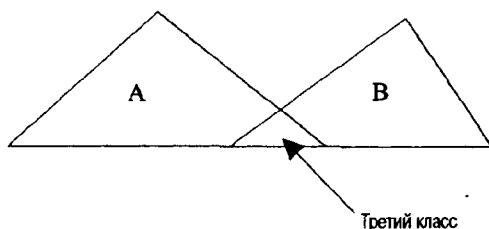


Рис. 2.25. Две треугольные области и их пересечение

13. Объясните, почему в сети с обратным распространением ошибок весовые коэффициенты инициализируются как положительными, так и отрицательными случайными значениями, а не только положительными.
14. Подумайте о том, чем отличается сеть с обратным распространением ошибок, в которой в качестве функций активности используются функции Гаусса, от сети с радиальными базисными функциями.

Глава 3

Кластеризация образцов

Задача. Описание обучения без управления.

Цели. Вы должны понять:

что такое обучение без управления и что такое принцип кластеризации образцов;
принципы построения самоорганизующейся сети Кохонена, чтобы иметь возможность реализовать такую сеть на том языке программирования, который вы предпочтете.

Требования. Знание основ линейной алгебры на уровне, представленном в приложении А. Знакомство с материалом главы 1.

3.1. Основные идеи

В предыдущей главе мы рассмотрели управляемое обучение, когда нейронная сеть обучается классифицировать образцы в соответствии с инструкциями: целевой выходной образец дает информации сети о том, к какому классу следует научиться относить входной образец. При обучении без управления таких инструкций нет, и сети приходится проводить кластеризацию образцов (т.е. разделение их на группы) самостоятельно. Все образцы одного кластера должны иметь что-то общее — они будут оцениваться, как подобные. Предположим, например, что перед нами стоит задача классификации мебели по признакам полезности и красоты. Все объекты, подобное стулу, попадут при этом в одну группу, а все объекты, подобные столу, — в другую. Эти группы затем анализируются, и от группы подобных столу предметов отделяется группа письменных столов. Группа письменных столов подобна группе предметов, подобных столам, поэтому эти группы должны разместиться близко одна к другой и далеко от группы предметов, подобных стулу. Алгоритмы кластеризации выполняют такие операции с образцами данных. Группы в дальнейшем мы будем называть *кластерами* и предполагать, что разделение образцов на кластеры должно удовлетворять следующим двум требованиям.

- Образцы внутри одного кластера должны быть в некотором смысле подобны.
- Кластеры, подобные в некотором смысле, должны размещаться близко один от другого.

На рис. 3.1 показано размещение на двумерной плоскости данных, которые естественным образом организуются в три кластера: соответствующая образцу точка попадает в определенный кластер, если она располагается близко к точкам этого кластера в сравнении с точками, принадлежащими другим кластерам. Мерой близости (или подобия) двух точек обычно является квадрат евклидова расстояния между ними, вычисляемый по формуле

$$d_{pq} = \sum_i^n (x_{pi} - x_{qi})^2,$$

где d_{pq} обозначает квадрат евклидова расстояния между точкой p и точкой q , x_{pi} — i -я координата образца p (аналогично для образца q), а n — значение размерности.

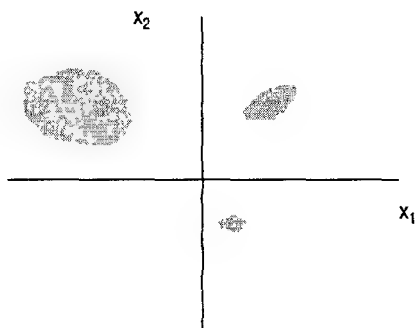


Рис. 3.1. Данные, формирующие три кластера

Если для кластера j рассмотреть вектор p_j , определяемый центроидом (точкой, соответствующей усредненной характеристике размещения всех образцов в кластере), то для данных на рис. 3.1 решение о том, какому из кластеров принадлежит произвольный вектор x (изображаемый на плоскости в виде точки), определяется значением

$$\text{index}(x) = \min d(p_j, x), \quad \text{для всех } j,$$

возвращающим индекс кластера с наименьшим квадратом евклидова расстояния до вектора x . Векторы p_j могут рассматриваться как прототипы кластеров, и эти прототипы могут служить для представления ключевых признаков кластера. Например, если необходимо разделить на группы

баскетболистов и жокеев, несомненно, что в качестве характерного признака можно выбрать рост. Поэтому значения элементов, означающих рост в векторах-прототипах двух кластеров, должны существенно отличаться.

Алгоритм кластеризации представляет собой статистическую процедуру выделения групп из имеющегося набора данных. Существует немало алгоритмов кластеризации самого разного уровня сложности. Один из самых простых подходов заключается в том, чтобы предположить существование определенного числа кластеров и произвольным образом выбрать координаты для каждого из прототипов. Затем каждый вектор из набора данных связывается с ближайшим к нему прототипом, и новыми прототипами становятся центроиды всех векторов, связанных с исходным прототипом. На рис. 3.2 показан случайный выбор прототипов, которые к концу обучения должны переместиться в центры кластеров, как показано на рис. 3.3.

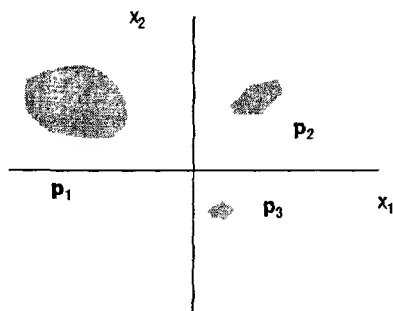


Рис. 3.2. Три случайных вектора, которые будут смещаться, выступая в роли прототипов для кластеров

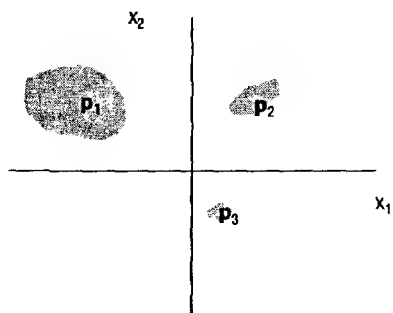


Рис. 3.3. Каждый из прототипов переместился в точку, соответствующую центроиду кластера

Пример 3.1

1. Найдите квадрат евклидового расстояния между векторами:

$$\mathbf{p} = [-2.3 \ 1.4],$$
$$\mathbf{x} = [4.5 \ 0.6].$$

2. Найдите квадрат евклидового расстояния между векторами:

$$\mathbf{p} = [0.4 \ 0.3 \ 1.1 \ 0.9],$$
$$\mathbf{x} = [0.6 \ 0.7 \ -0.5 \ 1.1].$$

Решение 3.1

1. $d(\mathbf{p}, \mathbf{x}) = (-2.3 - 4.5)^2 + (1.4 - 0.6)^2 = 46.9.$
2. $d(\mathbf{p}, \mathbf{x}) = (0.4 - 0.6)^2 + (0.3 - 0.7)^2 + (1.1 - -0.5)^2 + (0.9 - 1.1)^2 = 2.8.$

Прототипы можно рассматривать, как резюме экзаменуемого набора данных. Кластеры на рис. 3.1 имеют разные размеры, и самый большой из них имеет достаточно большую протяженность по обеим осям. Иногда бывает удобно представлять кластер несколькими прототипами, чтобы получить более детальную характеристику данных. Чтобы распознать кластеры, являющиеся частями большего кластера, необходимо знать положение всех прототипов друг относительно друга. Одной из проблем применения алгоритмов кластеризации является выбор оптимального числа кластеров. Если число кластеров выбрать слишком малым, могут быть упущены некоторые важные характеристики данных, а если кластеров окажется слишком много, то мы не получим никакой эффективной итоговой информации о данных (может даже случиться, что каждый образец создаст свой кластер). Можно сформулировать некоторые основные свойства идеального алгоритма кластеризации:

- автоматическое определение числа прототипов;
- сравнение прототипов;
- представление характерных признаков прототипа.

На практике первым из указанных свойств не обладает ни один из известных алгоритмов кластеризации. Нейронная сеть с обучением без управления, выполняющая кластеризацию, представляет собой самоорганизующуюся карту признаков, которую в начале 80-х годов предложил Кохонен (Kohonen).

3.2. Самоорганизующаяся карта признаков

Самоорганизующаяся карта признаков (сеть SOFM — Self-Organizing Feature Map, см. [Kohonen, 1990]) имеет набор входных элементов, число

которых соответствует размерности учебных векторов, и набор выходных элементов, которые служат в качестве прототипов. Базовая архитектура сети SOFM показана на рис. 3.4. Например, для данных, показанных на рис. 3.1, потребуется сеть с двумя входными и по крайней мере тремя выходными элементами, представляющими три кластера.

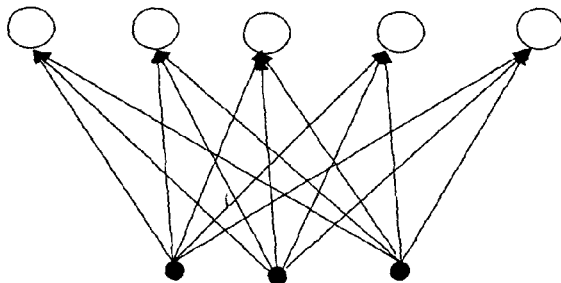


Рис. 3.4. Эта сеть имеет три входных и пять кластерных элементов, каждый элемент входного слоя связан с каждым элементом кластерного слоя

Входные элементы предназначены только для того, чтобы распределять данные входного вектора между выходными элементами сети. Выходные элементы называются *кластерными элементами*. Так как число входных элементов соответствует размерности вводимых векторов, а каждый входной элемент связан со всеми кластерными элементами, общее число влияющих на кластерный элемент весовых значений тоже оказывается равным размерности входных векторов. Часто удобно интерпретировать весовые значения кластерного элемента как значения координат, описывающих позицию кластера в пространстве входных данных. На рис. 3.5 показано, как связываются весовые значения с пространством входных данных.

Кластерные элементы размещаются в виде одно- или двумерного масштаба, как показано на рис. 3.6. В ходе обучения все элементы могут рассматриваться как претенденты на награды в виде учебных векторов. Когда на конкурс выставляется какой-либо учебный вектор, вычисляются расстояния от него до всех кластерных элементов, и элемент, который находится к данному учебному вектору ближе всех, объявляется элементом-победителем. Для элемента-победителя выполняется корректировка весовых значений так, чтобы этот кластерный элемент стал к учебному вектору еще ближе. Обычно корректировка весовых значе-

ний выполняется и для элементов, близких к элементу-победителю. Весовые значения элемента подлежат обновлению, если элемент лежит внутри круга заданного радиуса с центром в элементе-победителе. В ходе обучения радиус обычно постепенно уменьшается. Норма обучения ограничивает величину, на которую кластерный элемент может передвинуться по направлению к учебному вектору, и, подобно радиусу, норма обучения тоже со временем постепенно уменьшается. В наших примерах рассматривается размещение кластерных элементов в виде линейного массива или квадратной сетки, но могут использоваться и другие формы, например шестиугольная сетка. От топологии зависит только то, какие элементы должны обновляться для данного конкретного радиуса.

Обычно число кластерных элементов выбирают меньшим, чем число учебных образцов, поскольку целью является получение упрощенной характеристики данных. К концу обучения кластерные элементы обеспечивают “информационную сводку” по пространству входных образцов. Кластерные элементы выступают в роли карты признаков пространства входных данных. Например, немного позже в этой же главе мы увидим, что в результате кластеризации изображений символов разные области кластерных элементов будут характеризовать разные символы (или их комбинации).

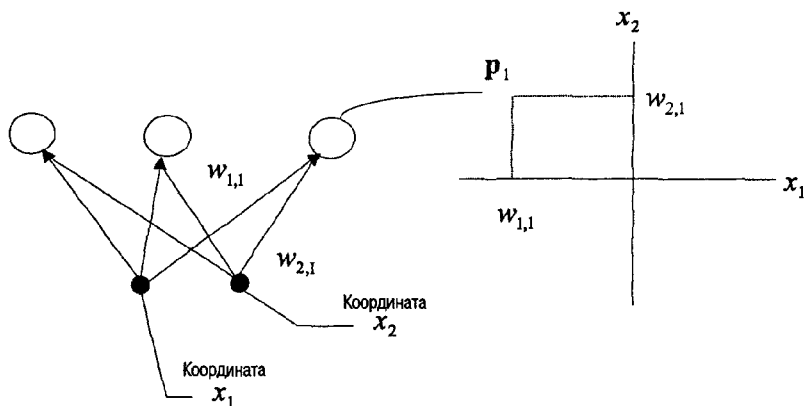
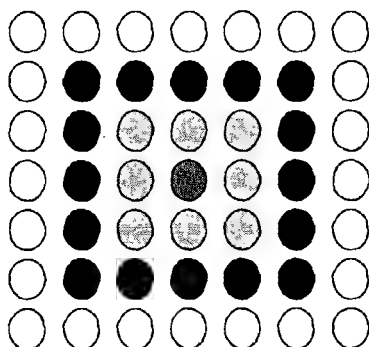
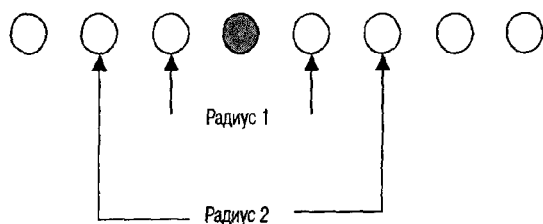


Рис. 3.5. На этой схеме показано, как в качестве прототипа может выступать кластерный элемент. Входной слой использует значения координат точки входного пространства. В процессе обучения весовые значения корректируются, и по окончании обучения все кластерные элементы займут в пространстве входных данных положение, соответствующее полученным для них весовым значениям



 Элемент-победитель – только он должен обновляться, когда радиус равен 0

  Элементы, которые должны обновляться, когда радиус равен 1

   Элементы, которые должны обновляться, когда радиус равен 2

Рис. 3.6. Вверху кластерные элементы сети размещены в линию, а внизу – в виде квадратной сетки. От размещения зависит, какие элементы попадут внутрь круга данного радиуса с центром в элементе-победителе. Элементы можно разместить иначе (например, в виде треугольника или шестиугольника), но чаще всего используются именно линейное размещение и размещение в виде квадрата

3.2.1. Алгоритм

В следующем алгоритме η обозначает норму обучения, а n — шаг во времени.

инициализировать весовые значения случайными значениями

выполнять, пока not HALT

для каждого входного вектора

для каждого кластерного элемента вычислить расстояние до учебного вектора:

$$d_j = \sum_i (w_{ij} - x_i)^2$$

найти элемент j , для которого расстояние минимально

для элементов из круга заданного радиуса обновить весовые векторы по формуле

$$w_{ij}(n+1) = w_{ij}(n) + \eta(n)[x_i - w_{ij}(n)]$$

проверить, требуется ли обновление нормы обучения или радиуса

проверить HALT

Учебные векторы выбираются из набора векторов случайным образом. Условие HALT (останов) выполняется тогда, когда величины изменения весов для всех кластерных элементов становятся очень маленькими — при этом условии учебные векторы должны попадать в одни и те же зоны карты при переходе от одной эпохи к следующей.

Норма обучения со временем меняется. Она может, например, сначала иметь значение 0.9, а затем уменьшаться линейно до некоторого фиксированного малого значения (скажем, 0.01), после чего оставаться неизменной. Радиус обычно выбирается достаточно большим, чтобы сначала обновлялись все элементы. Радиус тоже со временем уменьшается, и в конце, как правило, должны обновляться только несколько соседствующих с элементом-победителем элементов или вообще только сам этот элемент. Норма обучения тоже может зависеть от того, насколько близко размещается обновляемый элемент к элементу-победителю.

Пример 3.2

1. Сколько эпох потребуется для того, чтобы значение η уменьшилось до 0.1, при условии использования следующего правила обновления:

$$\eta(0) = 0.90,$$

$$\eta(n+1) = \eta(n) - 0.001.$$

2. Сеть SOFM имеет вид двумерной сетки размером 10×10 , а радиус изначально задан равным 6. Выясните, как много элемен-

тов должны будут обновляться после 1000 эпох, если элемент-победитель размещен в правом нижнем углу сетки, а радиус меняется по правилу:

$$r = r - 1, \text{ если номер_текущей_эпохи mod } 200 = 0.$$

Предполагается, что нумерация эпох начинается с 1.

Решение 3.2

1. $0.1 = 0.9 - n \cdot 0.001,$

$$\therefore n = \frac{0.9 - 0.1}{0.001} = 800.$$

2. Если предположить, что счет эпох начинается с 1, радиус будет уменьшаться на 1 после каждых 200 эпох. После 1000 эпох радиус окажется равным 1. Число изменяемых элементов будет равно четырем, включая элемент-победитель.

3.2.2. Обучение сети SOFM

Карта признаков проходит два этапа обучения. На первом этапе элементы упорядочиваются так, чтобы отражать пространство входных элементов, а на втором этапе происходит уточнение их позиций. Как правило, процесс представляется визуально путем использования двумерных данных и построения соответствующей поверхности. Например, входные векторы выбираются случайным образом на основе однородного распределения в некотором квадрате, и начинается обучение карты. В определенные моменты в ходе обучения строятся изображения карты путем использования соответствия, показанного на рис 3.5. Элементы соединяются линиями, чтобы показать их относительное размещение. Сначала карта выглядит сильно “измятой”, но постепенно в ходе обучения она разворачивается и расправляется. Конечным результатом обучения является карта, покрывающая все входное пространство и являющаяся достаточно регулярной (т.е. элементы оказываются распределенными почти равномерно). Для примера была рассмотрена карта с топологией квадрата из 49 элементов, и для 250 точек данных, взятых из единичного квадрата, было проведено ее обучение, которое начиналось со случайного набора весовых значений, задающих размещение кластерных элементов в центре входного пространства, как показано на рис. 3.7. На рис. 3.8 и 3.9 иллюстрируется процесс разворачивания карты с течением времени.

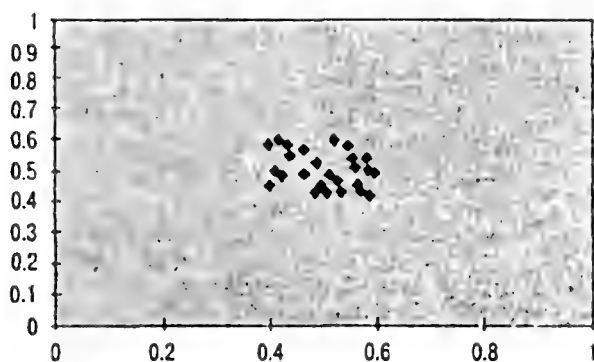


Рис. 3.7. Весовые векторы инициализируются случайными значениями из диапазона 0.4–0.6

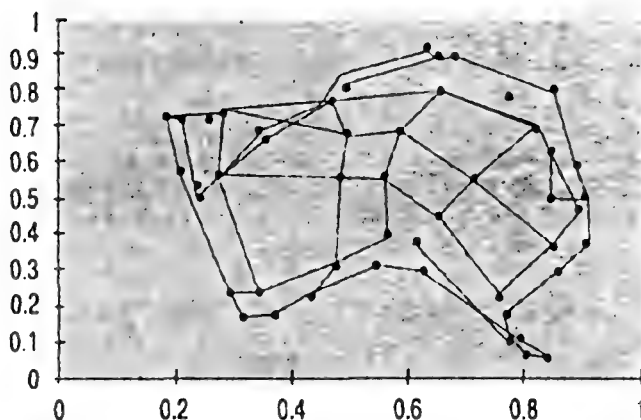


Рис. 3.8. Карта по прошествии 20 эпох

Как и для других типов сетей, в данном случае результат обучения зависит от учебных данных и выбора параметров обучения. На рис. 3.10 показано распределение 100 учебных образцов в единичном квадрате. На этом наборе данных была обучена карта с топологией квадрата из 25 элементов, а обучение начиналось со случайного набора весовых значений, соответствующих размещению элементов в центре пространства входных данных. Данная сеть относительно мала, поэтому совсем не удивительно, что конечным результатом использования неоднородно распределенных данных явилась нерегулярная карта, показанная на рис. 3.11.

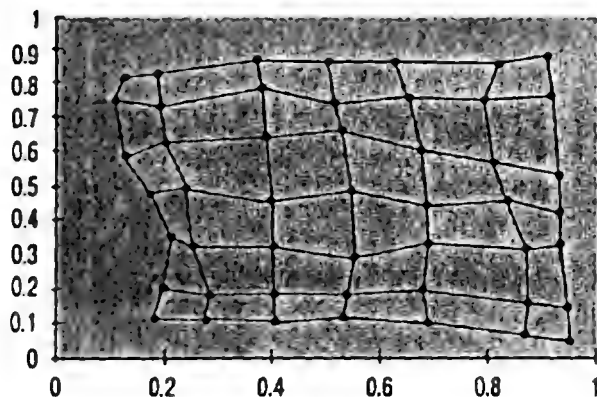


Рис. 3.9. Карта по прошествии 20 эпох незадолго до окончания обучения. Элементы теперь упорядочены, и карта станет еще более регулярной по окончании финальной фазы сходимости

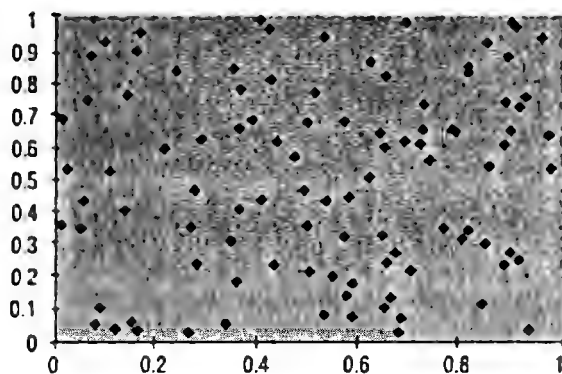


Рис. 3.10. Данные, сгенерированные случайным образом. Этот набор данных не однороден — здесь имеются пустоты, так что некоторые области входных данных оказываются представленными весьма бедно

Эта карта имеет “смятый” участок в верхнем правом углу. Скручивания могут быть результатом неправильного выбора начальных весовых значений, которые в начале обучения задают местоположение каждого кластерного элемента в пространстве входных данных. Например, если в начале

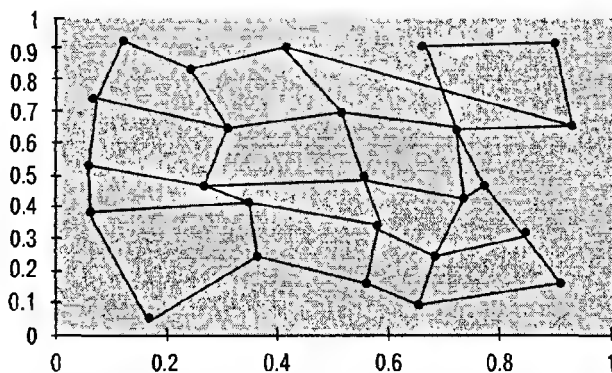


Рис. 3.11. Размещение элементов карты после обучения сети с использованием данных, показанных на рис. 3.10. Соседние элементы соединены линиями

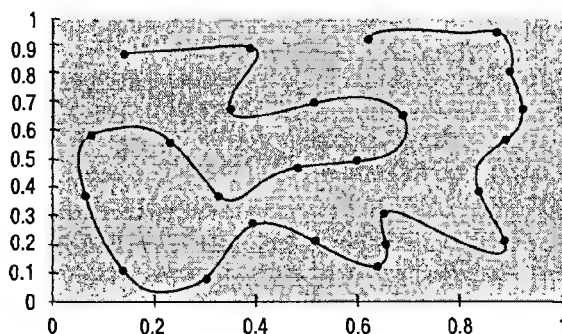


Рис. 3.12. Линейная карта из 25 элементов, соответствующих случайным данным из единичного квадрата

обучения все элементы взвешиваются так, что они оказываются смещенными в одном направлении в пространстве входных данных, то процесс упорядочения может усложниться. На рис. 3.12 показан тот же эксперимент, но на этот раз карта представляет собой линейный массив элементов. Этот линейный массив скручивается так, чтобы он заполнил пространство входных данных. Те же самые свойства пространственного заполнения можно применить и в случае других форм. Например, если набор учебных данных выбирается равномерно из треугольника, то можно разместить элементы квадратной карты так, чтобы эти элементы заполнили треугольник, как показано на рис. 3.13.

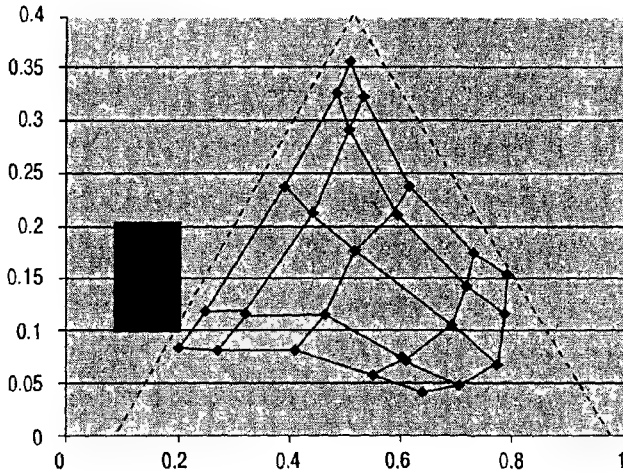


Рис. 3.13. Квадратная сетка сети SOFM, постепенно превращающаяся в треугольную. Учебные данные были взяты из закрашенной серым области, а весовые значения сети перед началом обучения выбирались случайным образом так, чтобы все элементы располагались внутри черного прямоугольника

Хехт-Нильсен [Hecht-Nielsen, 1990] предлагает подобный пример двумерного отображения, но в том примере задачей является построение карты угловых характеристик движения руки робота с координатами (x, y) . Описание соответствующего эксперимента приводится ниже.

Для того чтобы представить координаты (x, y) захвата робота, где соседние элементы представляются соседними координатами, будет использоваться карта квадратной формы. Можно представлять карту как шахматную доску (или ее часть, в зависимости от числа используемых элементов). Идея показана на рис. 3.14. Соотношение между углами и координатами задается следующими формулами:

$$x = l_1 \cos \theta_1 + l_2 \cos(\theta_1 + \theta_2),$$

$$y = l_1 \sin \theta_1 + l_2 \sin(\theta_1 + \theta_2),$$

$$\theta_2 = \cos^{-1} \left(\frac{x^2 + y^2 + l_1^2 + l_2^2}{2l_1 l_2} \right),$$

$$\theta_1 = \tan^{-1}(y/x) + \tan^{-1} \left(\frac{l_2 \sin \theta_2}{l_1 + l_2 \cos \theta_2} \right).$$

Это соотношение является нелинейным, но соседним координатам (x, y) будут соответствовать близкие углы.

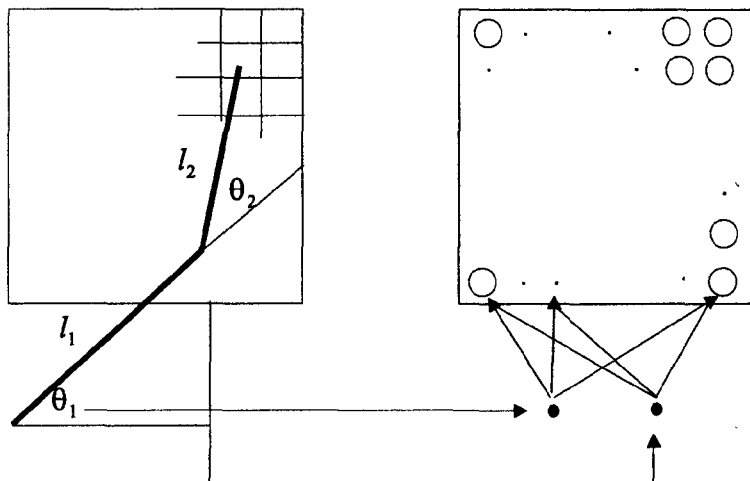


Рис. 3.14. Построение отображения углов, характеризующих сегменты руки, в узлы сетки. Если рука должна переместиться, например, в верхний правый угол, то значения соответствующего элементу верхнего правого угла весового вектора будут задаваться значениями углов, которые должны получить сегменты руки. Интерпретируйте это как давление на элемент в направлении, куда должен переместиться конец руки. Элемент в результате покажет соответствующие значения углов

Здесь представлены результаты одной попытки без какого-либо уточнения. Скорее всего после нескольких экспериментов можно получить и лучшие результаты. Рука робота была размещена так, как показано на рис. 3.15. Набор учебных данных был очень небольшим и содержал 25 точек: $\{(4, 1), (4, 2), (4, 3), (4, 4), (4, 5), (5, 1), (5, 2), (5, 3), (5, 4), (5, 5), \dots (8, 5)\}$. Были вычислены углы, соответствующие нахождению конца руки робота в этих точках, и значения этих углов использовались для обучения сети SOFM. Элементов обычно используют намного меньше, чем точек, соответствующих данным обучения, поскольку основное назначение сети SOFM — получение сводки о пространстве входных данных в терминах некоторого набора прототипов. Однако в данном эксперименте это не так, и здесь была рассмотрена квадратная карта из 49 элементов. Целью являлась демонстрация того, что в этом случае с каждым элементом может быть связано не более одной соответствующей учебным данным точки и выяснение того, как при этом распределяются элементы в простран-

стве входных данных. Сначала весовые значения устанавливаются равными малым случайным значениям. Результаты, полученные в ходе эксперимента, показаны на рис. 3.16–3.18. Из рис. 3.18 видно, что для каждого учебного образа имеется представляющий его элемент. Кроме того, имеется большое число свободных элементов, занимающих в пространстве входных данных промежуточные положения.

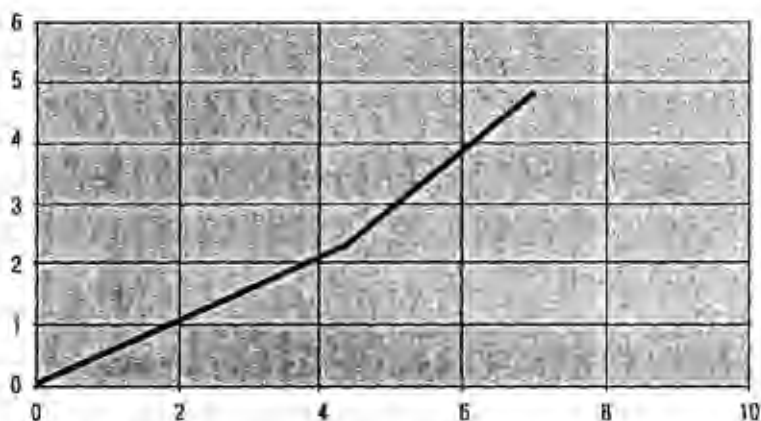


Рис. 3.15. Рука робота в пространстве, заданном сеткой

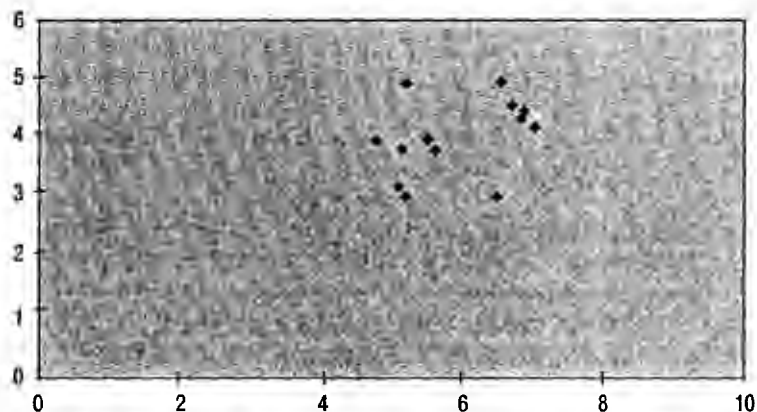


Рис. 3.16. Размещение весовых векторов в координатах (x, y) после 5000 циклов обработки всех учебных данных

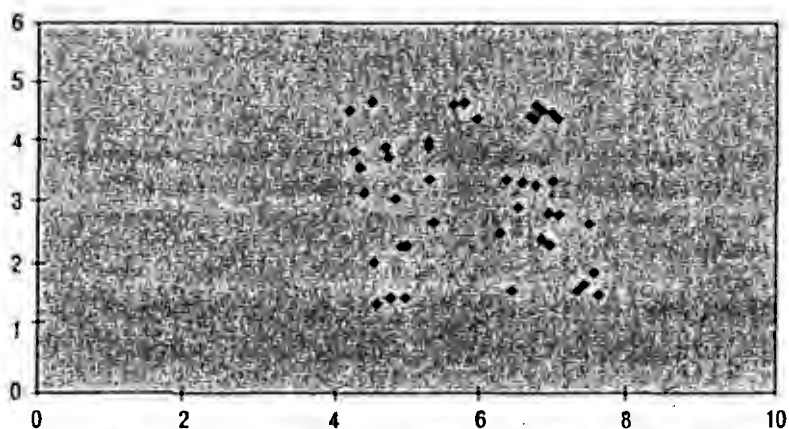


Рис. 3.17. Размещение весовых векторов в координатах (x, y) после 10000 циклов обработки всех учебных данных

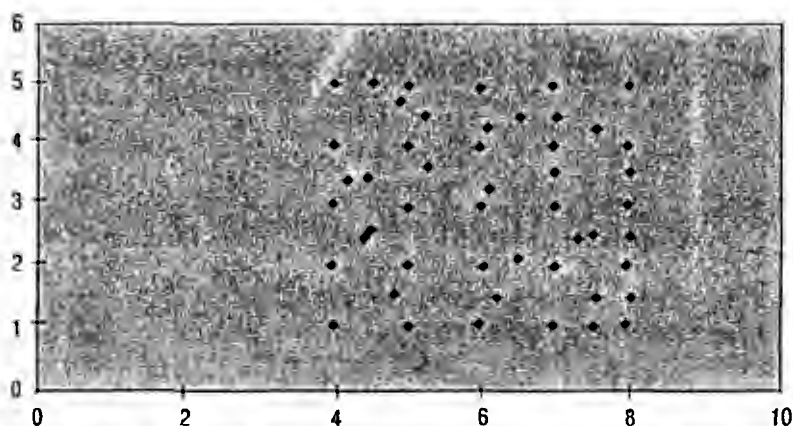


Рис. 3.18. Размещение весовых векторов в координатах (x, y) после 20000 циклов обработки всех учебных данных

Пример 3.3

Для обучения сети SOFM с тремя входными и двумя кластерными элементами используются четыре учебных вектора

$[0.8 \ 0.7 \ 0.4]$, $[0.6 \ 0.9 \ 0.9]$, $[0.3 \ 0.4 \ 0.1]$, $[0.1 \ 0.1 \ 0.3]$

и начальные весовые значения

$$\begin{bmatrix} 0.5 & 0.4 \\ 0.6 & 0.2 \\ 0.8 & 0.5 \end{bmatrix} :$$

Начальный радиус выбирается равным 0, а норма обучения η — равной 0.5. Вычислите изменения весовых значений в ходе первого цикла обработки данных, рассматривая учебные векторы в указанном порядке.

Решение 3.3

Рассматривая входной вектор 1, для кластерного элемента 1 получаем

$$d_1 = (0.5 - 0.8)^2 + (0.6 - 0.7)^2 + (0.8 - 0.4)^2 = 0.26 ,$$

а для кластерного элемента 2 —

$$d_2 = (0.4 - 0.8)^2 + (0.2 - 0.7)^2 + (0.5 - 0.4)^2 = 0.42 .$$

Элемент 1 оказывается ближе, поэтому

$$w_{ij}(n+1) = w_{ij}(n) + 0.5[x_i - w_{ij}(n)] .$$

Новыми весовыми значениями являются

$$\begin{bmatrix} 0.65 & 0.4 \\ 0.65 & 0.2 \\ 0.60 & 0.5 \end{bmatrix} .$$

Рассматривая входной вектор 2, для кластерного элемента 1 получим

$$d_1 = (0.65 - 0.6)^2 + (0.65 - 0.9)^2 + (0.60 - 0.9)^2 = 0.155 ,$$

а для кластерного элемента 2 —

$$d_2 = (0.4 - 0.6)^2 + (0.2 - 0.9)^2 + (0.5 - 0.9)^2 = 0.69 .$$

Элемент 2 оказывается ближе, и новыми весовыми значениями будут

$$\begin{bmatrix} 0.625 & 0.400 \\ 0.775 & 0.200 \\ 0.750 & 0.500 \end{bmatrix} .$$

Рассматривая входной вектор 3, для кластерного элемента 1 получим

$$d_1 = (0.625 - 0.3)^2 + (0.775 - 0.4)^2 + (0.75 - 0.1)^2 = 0.67 ,$$

а для кластерного элемента 2 —

$$d_2 = (0.4 - 0.3)^2 + (0.2 - 0.4)^2 + (0.5 - 0.1)^2 = 0.21 .$$

Элемент 1 оказывается ближе, и новыми весовыми значениями будут

$$\begin{bmatrix} 0.625 & 0.350 \\ 0.775 & 0.300 \\ 0.750 & 0.300 \end{bmatrix}.$$

Рассматривая входной вектор 4, для кластерного элемента 1 получим

$$d_1 = (0.625 - 0.1)^2 + (0.775 - 0.1)^2 + (0.75 - 0.3)^2 = 0.93,$$

а для кластерного элемента 2 —

$$d_2 = (0.35 - 0.10)^2 + (0.30 - 0.10)^2 + (0.30 - 0.30)^2 = 0.10.$$

Элемент 2 оказывается ближе, и новыми значениями весов будут

$$\begin{bmatrix} 0.625 & 0.225 \\ 0.775 & 0.200 \\ 0.750 & 0.300 \end{bmatrix}.$$

3.2.3. Дополнительные сведения о сети SOFM

В определенных случаях в качестве меры сходства векторов можно использовать угол между ними. Взгляните на рис. 3.19. Вектор $\mathbf{a}$ ближе к прототипу $\mathbf{p}_1$ в смысле евклидова расстояния, но ближе к прототипу $\mathbf{p}_2$, если в качестве меры сходства выбрать значение угла. *Произведение* (называемое также *скалярным произведением*) векторов

$$\mathbf{v} = [v_1 \ v_2 \ \dots \ v_n] \text{ и } \mathbf{w} = [w_1 \ w_2 \ \dots \ w_n]$$

определяется формулой

$$\mathbf{v} \cdot \mathbf{w} = [v_1 w_1 \ v_2 w_2 \ \dots \ v_n w_n].$$

Угол между ненулевыми векторами $\mathbf{v}$ и $\mathbf{w}$ равен

$$\cos^{-1} \left(\frac{\mathbf{v} \cdot \mathbf{w}}{\|\mathbf{v}\| \|\mathbf{w}\|} \right).$$

Норма или модуль вектора вычисляется по формуле $\|\mathbf{v}\| = \sqrt{v_1^2 + v_2^2 + \dots + v_n^2}$. Вектор можно нормализовать путем деления каждого элемента вектора на норму вектора. Если все векторы нормализованы, то индекс прототипа-победителя для входного образца задается условием:

$$\text{index}(\mathbf{x}) = \max \{ \mathbf{p}_j \cdot \mathbf{x} \} \quad \text{для всех } j.$$

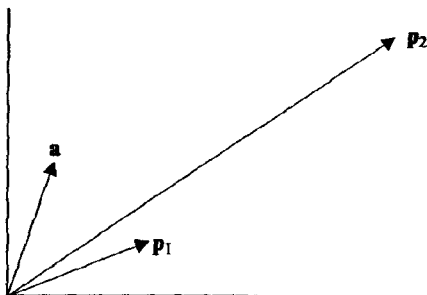


Рис. 3.19. Три вектора, используемые в примере 3.4

Обучение сети SOFM, в которой в качестве меры сходства используется скалярное произведение векторов, проводится так же, как было описано выше, но формула обновления весов теперь должна быть следующей:

$$w_j(n+1) = \frac{w_j(n) + \eta x}{\|w_j(n) + \eta x\|}.$$

Таким образом, для элемента-победителя к его весовому вектору добавляется часть этого вектора, а затем полученный в результате вектор нормализуется.

Пример 3.4

Пусть векторами на рис. 3.19 являются

$$a = [1 \ 4],$$

$$p_1 = [2 \ 1],$$

$$p_2 = [6 \ 6].$$

- (а) Используя скалярное произведение, покажите, что a ближе к p_2 , чем к p_1 .
- (б) Если p_2 оказывается прототипом-победителем в некоторой сети SOFM, то покажите, как должен двигаться вектор p_2 в предположении, что вектор a предлагается на рассмотрение сети два раза подряд (т.е. в случае, когда нет других учебных образцов). Считайте, что $\eta = 1$.

Решение 3.4

(а) Нормализованные скалярные произведения равны:

$$ap_1^T = \frac{(1 \times 2) + (4 \times 1)}{\sqrt{1^2 + 4^2} \sqrt{2^2 + 2^2}} = 0.651,$$

$$ap_2^T = \frac{(1 \times 6) + (4 \times 6)}{\sqrt{1^2 + 4^2} \sqrt{6^2 + 6^2}} = 0.857.$$

Так что p_2 оказывается ближе. Обратите внимание на то, что ap_j^T используется здесь для обозначения нормализованного произведения, где верхний индекс означает транспонирование вектора.

(б) Нормализованными векторами являются

$$a = \begin{bmatrix} 1/\sqrt{17} & 4/\sqrt{17} \end{bmatrix},$$

$$p_2 = \begin{bmatrix} 6/\sqrt{72} & 6/\sqrt{72} \end{bmatrix}.$$

Для первого представления имеем:

$$p_2(n+1) = \frac{\begin{bmatrix} 6/\sqrt{72} + 1/\sqrt{17} & 6/\sqrt{72} + 4/\sqrt{17} \end{bmatrix}}{\sqrt{\left(6/\sqrt{72} + 1/\sqrt{17}\right)^2 + \left(6/\sqrt{72} + 4/\sqrt{17}\right)^2}} \\ = \{0.493 \quad 0.870\}.$$

Для второго представления имеем:

$$p_2(n+2) = \frac{\begin{bmatrix} 0.493 + 1/\sqrt{17} & 0.870 + 4/\sqrt{17} \end{bmatrix}}{\sqrt{\left(0.493 + 1/\sqrt{17}\right)^2 + \left(0.870 + 4/\sqrt{17}\right)^2}} \\ = \{0.371 \quad 0.928\}.$$

В работе [Kohonen, 1990] дается несколько практических советов для обучения сети SOFM. В течение первых 1000 итераций норма обучения должна быть около единицы, а после этого она должна постепенно снижаться. Характер уменьшения нормы обучения не имеет особого значения и может быть линейным, экспоненциальным или обратно пропорциональным числу итераций. Упорядочение карты признаков происходит на протяжении этой начальной фазы. После окончания фазы упорядочения норма обучения должна поддерживаться постоянной и малой (например, равной 0.1 или меньшему значению) на протяжении доста-

точно долгого времени, пока происходит уточнение карты. Радиус должен начинаться с больших значений (например, со значения, большего половины диаметра карты) и уменьшаться линейно в течение первых 1000 итераций, после чего радиус может оставаться равным 1 или 0. Число итераций должно быть большим, обычно между 10000 и 100000.

3.3. Эксперимент

В главе 2 был рассмотрен процесс настройки сети с прямой связью на основе использования управляемого обучения для распознавания четырех символов {A, B, C, D} из трех различных шрифтов. Эти учебные символы показаны снова на рис. 3.20. Сейчас мы рассмотрим еще один эксперимент с использованием тех же учебных образцов.

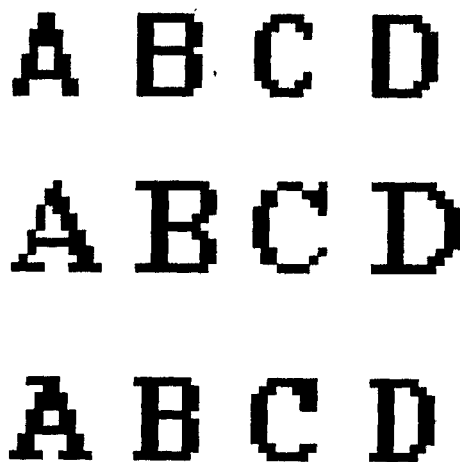


Рис. 3.20. Каждый ряд символов представляет свой шрифт

Целью этого эксперимента является демонстрация того, как сеть SOFM группирует учебные экземпляры данных. Для демонстрации была выбрана квадратная сетка из девяти элементов. Начальное значение радиуса было установлено равным 3, а норма обучения — равной 1. Сеть обучалась в течение 3000 циклов обработки всех образцов. В табл. 3.1 показано, как группируются символы по окончании обучения. Каждая ячейка таблицы соответствует одному элементу сетки.

Таблица 3.1. Элементы сети SOFM с приписанными им символами: разные числа соответствуют разным шрифтам

A1 A2 A3		B1 D1 B3 D3
B2 D2		C1 C2 C3

Изображения символов задаются сеткой 16×16 пикселей, так что для каждого элемента получается 256 входных значений. Мы считаем, что после достижения сходимости в сети, каждый из элементов можно будет считать прототипом для тех образцов, которые окажутся с ним связанными. Если представить весовые векторы в виде изображений, можно ожидать, что изображения элементов будут выглядеть подобно изображениям образцов, для которых этот элемент окажется прототипом. Процедура построения изображения элемента довольно проста. В оригинальном поточечном изображении символа белые пиксели представлены значением 0, а черные — значением 1. Поэтому все весовые значения должны быть между 0 и 1. Каждое весовое значение связывается с одним пикселем оригинальной сетки, поэтому каждый вес можно интерпретировать как значение, характеризующее оттенок серого цвета (в действительности оттенки серого характеризуются значениями от 0 до 255, поэтому и весовые значения придется масштабировать в этом диапазоне). На рис. 3.21, где показаны изображения наборов весовых значений каждого из элементов, можно видеть, с каким именно элементом вероятнее всего следует связать образец. В этом эксперименте все символы “А” были сгруппированы вместе, как и все символы “С”. Символы “В” и “D” имеют похожие формы, поэтому они тоже оказались сгруппированными вместе. В идеале можно было бы надеяться, что “В” и “D” тоже будут отнесены к разным кластерам. Однако в реальных задачах кластеризация указывает лишь на подобие и совсем не означает полную защиту от ошибок. Кроме того, мы сами при распределении объектов по категориям делаем это в зависимости от контекста. Например, корешки книг можно рассматривать как удобный информационный элемент, если иметь в виду организацию работы, или как украшения офиса, если иметь в виду дизайн. В данном примере сеть сообщила нам кое-что о символах,

не получив об этих символах никакой информации от нас (сеть обучалась без управления). Сеть сообщила нам, что шесть образцов (“А” и “С”) попали в две очень несходные группы. Конечно, воспользовавшись своим совершенным зрением, мы могли бы в данном случае распознать сходство разных форм быстрее и лучше, но суть здесь, однако, в том, чтобы иметь возможность анализировать данные, которые могут иметь очень много признаков и не могут быть представлены так просто. Если мы не знаем, как можно было бы классифицировать такие данные, сеть типа самоорганизующейся карты Кохонена может оказаться весьма полезной.

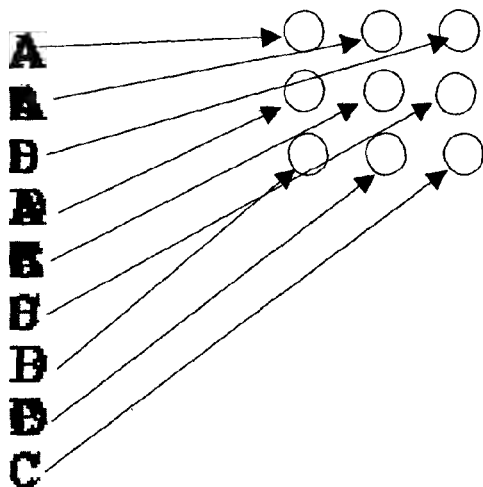


Рис. 3.21. Весовые значения элементов, представленные в виде изображений в оттенках серого

3.4. Резюме

Нейронная сеть может осуществлять группирование данных, используя процесс, напоминающий конкуренцию. Элементы сети могут также демонстрировать самоорганизацию, при которой элементы сети, представляемые в пространстве входных данных, повторяют топологию учебных данных. В качестве степени сходства обычно используются либо евклидово расстояние между векторами, либо угол между ними.

Самоорганизующаяся карта признаков имеет следующие свойства.

- Все кластерные элементы связываются с входными элементами. Задачей входных элементов является распределение вводимых признаков между кластерными элементами.
- В ходе обучения элементы конкурируют за образцы. Элемент-победитель, который является элементом, находящимся к входному образцу ближе всех других, корректирует значения своих весовых коэффициентов так, чтобы он стал к данному входному образцу еще ближе (т.е. чтобы он стал еще более сходным с ним).
- Обучение состоит из двух этапов. На первом этапе обновление весовых значений происходит и для всех элементов из некоторой окрестности элемента-победителя. Эта окрестность постепенно уменьшается. На втором этапе обучения весовые значения корректируются малыми добавками до тех пор, пока не будет достигнута сходимость сети.
- Прощедшая обучение сеть может использоваться для классификации неизвестных образцов на основе их сходства с образцами, предъявленными сети в процессе обучения.

3.5. Дополнительная литература

В работе [Kohonen, 1990] приводятся некоторые интересные примеры процесса самоорганизации и практические советы по использованию сетей SOFM. Статья Кохонена является также хорошим источником ссылок на публикации о других исследованиях, посвященных сетям SOFM. В [Hecht-Nielsen, 1990] содержится хороший обзор примеров конкурентного обучения. Другим широко используемым типом осуществляющих кластеризацию сетей являются сети, в которых применяется метод адаптивного резонанса. Они были предложены в [Carpenter, Grossberg, 1987], но для первого знакомства мы рекомендуем [Fausett, 1994].

3.6. Упражнения

1. Векторы x , p_1 и p_2 являются следующими:

$$x = [0.2 \quad -1.4 \quad 2.3],$$

$$p_1 = [0.6 \quad -4.0 \quad 7.0],$$

$$p_2 = [0.1 \quad -1.0 \quad 2.2].$$

- (а) К какому из прототипов оказывается ближе всего вектор x в смысле евклидова расстояния?
 - (б) К какому из прототипов оказывается ближе всего вектор x в смысле скалярного произведения?
 - (в) Скорректируйте весовой вектор прототипа-победителя из п. (а) в соответствии с алгоритмом обучения SOFM при норме обучения 0.8.
 - (г) Скорректируйте весовой вектор прототипа-победителя из п. (а) в соответствии с алгоритмом обучения SOFM для случая использования скалярного произведения при норме обучения 0.8.
2. Повторите вычисления упражнения 1 для следующих векторов:

$$x = [0.2 \quad -1.4 \quad -0.3 \quad 0.8],$$

$$p_1 = [0.3 \quad -3.0 \quad 1.0 \quad 0.2],$$

$$p_2 = [0.4 \quad -1.4 \quad -2.0 \quad 3.0].$$

3. Норма обучения в сети SOFM уменьшается в течение первых 1000 итераций по закону

$$\eta(n) = 0.15 \left(1 - \frac{n}{1000} \right),$$

где n обозначает номер итерации.

- (а) Сколько итераций будет выполнено прежде, чем норма обучения уменьшится до значения 0.003?
 - (б) Почему указанный закон не является хорошим выбором?
4. На рис. 3.22 показаны два кластера данных (на рисунке они закрашены). Не закрашенная область в верхнем левом квадранте содержит несколько прототипов (единичных весовых векторов), выступающих в качестве начальных состояний для сети SOFM. Принимая радиус равным нулю, объясните, что может происходить с картой признаков, если начать обучение сети с этими данными.
5. Все векторы на рис. 3.23 лежат на единичной окружности. Предполагая, что векторы всегда должны приводиться к такому виду, будет ли разница в оценке прототипа-победителя при использовании в качестве меры близости евклидова расстояния и скалярного произведения? Обоснуйте свое заключение.

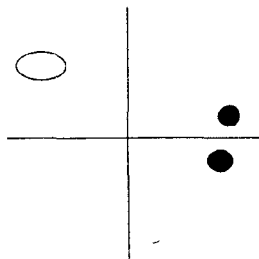


Рис. 3.22. Два кластера данных и прототипы для сети SOFM

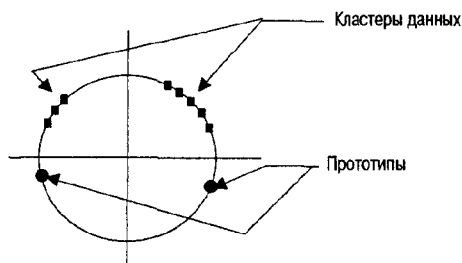


Рис. 3.23. Векторы на единичной окружности

6. Покажите, что для произвольного вектора x нахождение элемента-победителя по критерию минимума евклидова расстояния эквивалентно использованию критерия максимума скалярного произведения для нормализованных векторов. (Подсказка. В евклидовой метрике элемент-победитель может быть выражен через $\min \{ \|x - p_j\| \}$. Раскройте это выражение и сравните с нормализованным определением скалярного произведения.)
7. Линейная сеть SOFM с тремя элементами обучается решению проблемы XOR с использованием 0.1 и 0.9 для значений 0 и 1 соответственно и с использованием соответствующих четырех учебных образцов. При обучении использовались скалярные произведения и были получены указанные ниже окончательные весовые значения. Как были разделены на кластеры образцы?

Элемент 1	Элемент 2	Элемент 3
0.110	0.707	0.994
0.994	0.707	0.110

8. Сеть MAXNET (см. [Lippmann, 1987]) представляет собой конкурентную нейронную сеть, которая может использоваться для нахождения элемента сети, ввод которого оказывается максимальным. Каждый элемент соединен с самим собой, а также со всеми другими элементами двунаправленными взвешенными связями. Все весовые значения устанавливаются равными между собой, кроме веса автосвязи, который устанавливается равным 1:

$$w_{ij} = \begin{cases} 1, & \text{если } i = j, \\ -\omega, & \text{если } i \neq j, \end{cases}$$

где $0 < \omega < 1/N$, а N равно числу элементов сети.

Значение активности элемента устанавливается равным вводу, если ввод оказывается больше нуля, а иначе значение активности устанавливается равным нулю. Элемент получает взвешенный сигнал от всех других элементов и от себя самого. Элемент не изменяет свою активность до завершения итерации. Обновляются все элементы, и обновление происходит до тех пор, пока останется не более одного элемента с ненулевой активностью.

На рис. 3.24 показаны значения активности трех элементов при условии, что значения весов были инициализированы в соответствии с вышеуказанными правилами.

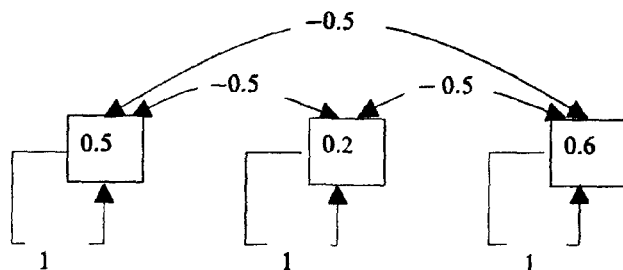


Рис. 3.24. Значения активности трех элементов

Для первой итерации вводом первого элемента будет $0.5 + 0.2 \times -0.5 + 0.6 \times -0.5 = 0.1$. Новым значением активности будет 0.1, поскольку ввод оказывается больше нуля. До конца итерации активность элемента будет оставаться равной 0.5. Ввод для второго элемента равен $0.2 + 0.5 \times -0.5 + 0.6 \times -0.5 = -0.35$. Новым значением активности будет 0, так как ввод оказывается меньше нуля. До конца первой итерации ак-

тивность элемента будет оставаться равной 0.2. Ввод для третьего элемента равен $0.6 + 0.5 \times -0.5 + 0.2 \times -0.5 = 0.25$. Новым значением активности будет 0.25, поскольку ввод оказывается больше нуля. В данный момент все элементы могут изменить значения активности.

Для второй итерации вводом первого элемента будет $0.1 + 0 + 0.25 \times -0.5 = -0.025$. Поэтому новым значением активности в конце итерации будет нуль. Ввод для второго элемента должен быть меньше нуля. Ввод для третьего элемента равен $0.25 + 0.1 \times -0.5 + 0 = 0.2$. Единственным элементом с ненулевой активностью оказывается третий элемент. Поэтому процесс завершится, и победителем будет третий элемент.

Повторите вышеприведенные рассуждения для значений активности элементов, равных 0.7, 0.6 и 0.3, соответственно.

9. Покажите, как можно использовать сеть MAXNET для нахождения элемента-победителя при решении упражнения 8.
10. Рассмотрите сеть MAXNET, состоящую из восьми элементов. Постройте схему сети и обозначьте на этой схеме соответствующие весовые значения. Покажите, что для выбранного вами набора значений активности сеть будет работать надлежащим образом.
11. Если для сети SOFM используется шестиугольная сетка и предполагается, что элементом-победителем является центральный элемент сетки, то сколько элементов должно быть обновлено при значении радиуса, равном 1, и сколько элементов должно быть обновлено при значении радиуса, равном 2?

Глава 4

Ассоциация образцов

Задача. Описание ассоциативной памяти.

Цели. Вы должны понять:

разницу между автоассоциативной и гетероассоциативной сетями;
принципы построения дискретной сети Хопфилда и двунаправленной автоассоциативной сети, чтобы иметь возможность реализовать эти сети с помощью программы электронных таблиц или на том языке программирования, который вы предпочтете;
как использовать для автоассоциации сеть с обратным распространением ошибок;
как обеспечить сжатие данных с помощью автоассоциативной сети с обратным распространением ошибок.

Требования. Знание основ линейной алгебры на уровне, представленном в приложении А. Знакомство с материалом главы 1.

4.1. Введение

Все знают, что такое ассоциация: слово одного языка может ассоциироваться со словом другого языка, фотография друга ассоциируется с его именем и можно даже умудриться ассоциировать некоторое туманное изображение с определенным реальным объектом. В главе 2 уже была рассмотрена идея ассоциативного обучения в виде, когда для каждого учебного образца имелся желаемый выходной образец. В этой главе мы рассмотрим случай запоминаемых пар образцов. Идея заключается в том, чтобы выбрать нужный образец из памяти, даже если у нас нет всей необходимой информации для начала поиска сохраненного образца. Например, вы хотите найти книгу в библиотеке, но не помните ее названия. При этом если вы знаете имя автора и описание того, чему книга посвящена, этого уже достаточно (с большой долей уверенности!), чтобы найти ассоциируемый с этой информацией объект.

Когда сохраняемая в памяти пара ассоциируемых образцов создается одинаковыми образцами, память называется *автоассоциативной*, а если образцы являются разными, то память называется *гетероассоциативной*. В этой главе будут рассмотрены три модели нейронных сетей для автоассоциации образов.

4.2. Дискретная сеть Хопфилда

Сеть Хопфилда (Hopfield) является автоассоциативной сетью, ведущей себя подобно памяти, которая может вспомнить сохраненный образец даже по подсказке (в виде вводимых данных), представляющей собой искаженную помехами версию нужного образца. Например, сеть может сохранить набор изображений букв, а когда сети будет представлена искаженная версия сохраненного символа, сеть должна оказаться способной найти истинный экземпляр. Дискретная сеть Хопфилда имеет следующие характеристики.

- Один слой элементов (входные элементы, представляющие входной образец, не учитываются).
- Каждый элемент связывается со всеми другими элементами, но элемент не связывается с самим собой.
- За один шаг обновляется только один элемент, в отличие, например, от сети с обратным распространением ошибок, где все элементы слоя могут изменяться одновременно, если сеть реализована в виде аппаратных средств с соответствующими параллельными возможностями.
- Элементы обновляются в случайном порядке, но в среднем каждый элемент должен обновляться в одной и той же мере. Например, в случае сети из 10 элементов после 100 обновлений каждый элемент должен обновиться приблизительно 10 раз.
- Вывод элемента ограничен значениями 0 или 1.

Сеть Хопфилда является рекуррентной в том смысле, что для каждого входного образца выход сети повторно используется в качестве ввода до тех пор, пока не будет достигнуто устойчивое состояние. Пример сети Хопфилда показан на рис. 4.1. Удобно считать, что сеть Хопфилда не имеет входных элементов, так как входной вектор просто определяет начальные значения активности элементов. Например, если ввод является двоичным, то входной вектор $[1\ 1\ 0\ 1]$ означает, что значения активности для элементов $\{1, 2, 4\}$ будут равны 1, а для элемента $\{3\}$ активность будет равна 0. Элемент обновляется тогда, когда все элементы передадут свои значения активности по имеющимся взвешенным связям, после чего вычисляется сумма произведений (т.е. бе-

рется скалярное произведение). Значение активности элемента получается на основе использования некоторого правила активизации. Каждый элемент сети Хопфилда имеет состояние, характеризующееся значением активности, которое должен посылать данный элемент другим элементам, а состояние сети в любой момент времени задается вектором состояний всех ее элементов.

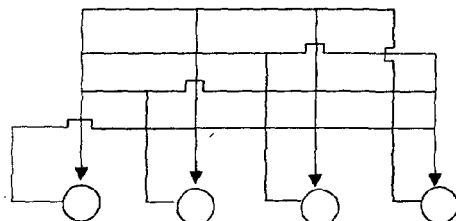


Рис. 4.1. Сеть Хопфилда с четырьмя элементами. Для каждого элемента входного вектора имеется свой элемент сети. Элементы сети связаны со всеми остальными ее элементами, но не сами с собой. Связи являются двунаправленными

В качестве входных данных сети Хопфилда можно использовать двоичные, но здесь мы будем использовать +1 для обозначения состояния “включено” и -1 — для состояния “выключено”. Комбинированный ввод элемента вычисляется по формуле:

$$net_i = \sum_{j=1}^n s_j w_{ij},$$

где s_j обозначает состояние элемента с номером j . Когда элемент обновляется, его состояние изменяется в соответствии с правилом

$$s_j = \begin{cases} +1, & \text{если } net_j > 0, \\ -1, & \text{если } net_j < 0. \end{cases}$$

Эта зависимость называется сигнум-функцией и в более краткой форме она записывается в виде

$$s_j = \text{sgn}(net_j).$$

Если комбинированный ввод оказывается равным нулю, то элемент остается в состоянии, в котором он пребывал перед обновлением.

Сеть работает очень просто. Входной вектор задает начальные состояния всех элементов. Элемент для обновления выбирается случайным образом. Выбранный элемент получает взвешенные сигналы от всех ос-

тальных элементов и изменяет свое состояние. Выбирается другой элемент, и процесс повторяется. Сеть достигает предела, когда ни один из ее элементов, будучи выбранным для обновления, не меняет своего состояния.

Весовые значения для сети Хопфилда определяются непосредственно из учебных данных без необходимости проведения обучения в более привычном смысле. Сеть Хопфилда ведет себя как память, и процедура сохранения отдельного вектора представляет собой вычисление прямого произведения вектора с ним самим. В результате этой процедуры создается матрица, задающая весовые значения для сети Хопфилда, в которой все диагональные элементы должны быть установлены равными нулю (поскольку диагональные элементы задают автосвязи элементов, а элементы сами с собой не связаны). Таким образом, весовая матрица, соответствующая сохранению вектора x , задается формулой

$$W = x^T x.$$

Пример 4.1.

Найдите набор весовых значений сети Хопфилда, соответствующий сохранению образца

$$[1 \ -1 \ 1 \ 1].$$

Решение 4.1.

$$\begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \end{bmatrix} [1 \ -1 \ 1 \ 1] = \begin{bmatrix} 1 & -1 & 1 & 1 \\ -1 & 1 & -1 & -1 \\ 1 & -1 & 1 & 1 \\ 1 & -1 & 1 & 1 \end{bmatrix}.$$

Поэтому весовыми значениями будут

$$W = \begin{bmatrix} 0 & -1 & 1 & 1 \\ -1 & 0 & -1 & -1 \\ 1 & -1 & 0 & 1 \\ 1 & -1 & 1 & 0 \end{bmatrix}.$$

Первый столбец представляет весовые значения, связанные с первым элементом, столбец 2 представляет весовые значения, связанные со вторым элементом, и т.д. Если сети будет предложен образец $[1 \ -1 \ 1 \ 1]$, то все элементы после обновления останутся в том же состоянии. Данные подсказки определяют начальные состояния всех элементов, так что в нашем случае второй элемент должен сначала находиться в состоянии -1 , а все остальные — в состоянии 1 . Пер-

вый элемент обновляется с помощью умножения вектора подсказки на первый столбец матрицы весов:

$$\begin{bmatrix} 1 & -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ -1 \\ 1 \\ 1 \end{bmatrix} = 3, \quad \text{sgn}(3) = 1.$$

Так что первый элемент останется в том же состоянии. Точно также при обновлении оставались бы неизменными и состояния всех остальных элементов.

Пример 4.2.

Найдите устойчивое состояние сети Хопфилда из примера 4.1 при условии, что входным образом является

$$\begin{bmatrix} -1 & -1 & 1 & 1 \end{bmatrix}.$$

Решение 4.2.

Элементы должны обновляться в случайном порядке. Для иллюстрации будем обновлять элементы в порядке 3, 4, 1, 2. Сначала рассмотрим элемент 3:

$$\begin{bmatrix} -1 & -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 0 \\ 1 \end{bmatrix} = 1, \quad \text{sgn}(1) = 1.$$

Таким образом, элемент 3 остается в том же состоянии. Следующим является элемент 4:

$$\begin{bmatrix} -1 & -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 1 \\ 0 \end{bmatrix} = 1, \quad \text{sgn}(1) = 1.$$

Так что элемент 4 остается в том же состоянии. Теперь элемент 1:

$$\begin{bmatrix} -1 & -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ -1 \\ 1 \\ 1 \end{bmatrix} = 3, \quad \text{sgn}(3) = 1.$$

Так что элемент 1 изменит свое состояние с -1 на 1 . Наконец, элемент 2:

$$[1 \ -1 \ 1 \ 1] \begin{bmatrix} -1 \\ 0 \\ -1 \\ -1 \end{bmatrix} = -3, \quad \text{sgn}(-3) = -1.$$

Так что элемент 2 останется в том же состоянии. Мы видим, что выявился ранее сохраненный вектор, характеризующий устойчивое состояние сети. Чтобы убедиться в том, что это состояние на самом деле является устойчивым, необходимо проверить, что в результате обновления ни один из элементов действительно не изменит своего состояния.

Процедура сохранения нескольких образов в сети Хопфилда тоже проста: прямое произведение вычисляется для каждого вектора, и все полученные таким образом весовые матрицы складываются.

Пример 4.3.

Определите весовую матрицу сети Хопфилда, соответствующую сохранению следующих двух векторов

$$\begin{bmatrix} -1 & 1 & -1 \end{bmatrix}, \\ \begin{bmatrix} 1 & -1 & 1 \end{bmatrix}.$$

Решение 4.3.

Соответствующей весовой матрицей является матрица

$$\begin{aligned} \mathbf{W} &= \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} \begin{bmatrix} -1 & 1 & -1 \end{bmatrix} + \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & -1 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 0 & -2 & 2 \\ -2 & 0 & -2 \\ 2 & -2 & 0 \end{bmatrix}. \end{aligned}$$

Диагональные элементы были обнулены.

Эксперимент

Сеть Хопфилда использовалась для того, чтобы сохранить три образца, изображения которых показаны на рис. 4.2. Каждый образец рассматривался в виде строки биполярных значений, где 1 представляла черный цвет, а -1 — серый. Строка состояла из 63 элементов, поэтому сеть имела 63 элемента и $63 \times 63 = 63$ весовых значений (это число весовых значений прямого произведения минус нулевые диагональные элементы). После определения весовых значений сети была предъявлена искаженная помехами версия числа "1". Образец, который выявился после некоторого числа итераций, показан на рис. 4.3.

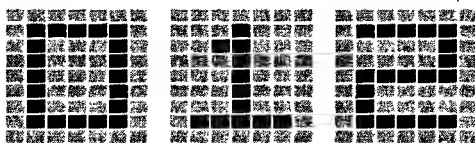


Рис. 4.2. Три образца, сохраняемые сетью Хопфилда. Черным пикселям соответствуют значения 1, а серым — значения -1

Этот эксперимент демонстрирует правильное восстановление сохраненного образца по искаженной его версии. Иногда сеть Хопфилда допускает ошибки, например, выявляя неверный образец или показывая искаженную версию правильного образца (в последнем случае выявленный образец может быть распознан, но все еще искажен).

В ряде работ имеются оценки, касающиеся числа образцов, которые можно сохранить в сети Хопфилда. Одной такой общей оценкой из [Haykin, 1994] является оценка

$$P_{\max} = \frac{N}{2 \ln N},$$

характеризующая максимальное число образцов, которые могут быть сохранены, если требовать правильного распознавания большинства образов; N обозначает число элементов сети.

4.2.1. Функция энергии

В работе [Hopfield, 1984] Хопфилд доказал, что его сеть должна сходиться к устойчивому набору значений активности, рассмотрев функцию энергии системы. В представленной нами форме сеть определяет необходимость изменения состояния элемента по пороговому значению, равному

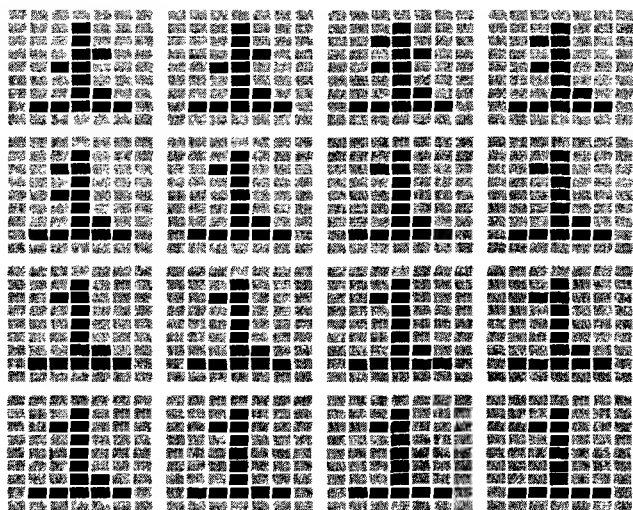


Рис. 4.3. Соответствующий вводимой подсказке образец показан в первом кадре в верхнем левом углу. Кадры представляют десять итераций сети (т.е. десять случайных обновлений). Кадры следует рассматривать в порядке слева направо и сверху вниз

нулю, поэтому мы можем использовать упрощенную версию функции энергии Хопфилда, подобную той, которая приводится в [Найкин, 1994]:

$$E = -\frac{1}{2} \sum_j \sum_i s_j s_i w_{ij}.$$

Если элемент j изменяет свое состояние на Δs_j , то изменение энергии будет равно

$$\Delta E = -\Delta s_j \sum_i s_i w_{ij}.$$

Можно рассматривать изменение энергии как функцию Δs_j и $\sum_i s_i w_{ij}$. Из табл. 4.1 видно, что Δs_j и $\sum_i s_i w_{ij}$ имеют одинаковые знаки, поэтому энергия при переходе от итерации к итерации всегда уменьшается. Например, первая строка данных таблицы говорит о том, что если элемент находится в положительном состоянии и его комбинированный ввод больше нуля (т.е. положителен), то новое состояние останется положительным и изменение состояния тоже будет положительным, а изменение энергии таким образом будет отрицательным.

Таблица 4.1. Изменения энергии при изменениях состояния элемента

Старое s_j	$\sum s_i w_{ij}$	Новое s_j	Δs_j	Изменение энергии
Положительное	Положительное	Положительное	Положительное	Отрицательное
Положительное	Отрицательное	Отрицательное	Отрицательное	Отрицательное
Отрицательное	Отрицательное	Отрицательное	Отрицательное	Отрицательное
Отрицательное	Положительное	Положительное	Положительное	Отрицательное

4.3. Двухнаправленная ассоциативная память

Сетью, имеющей много общего с сетью Хопфилда, является двухнаправленная ассоциативная память (сеть ВАРМ — Bidirectional Associative Memory), предложенная Коско (см. [Kosko, 1988]). Сеть ВАРМ является гетероассоциативной рекуррентной сетью. Сеть сохраняет пары образов и может восстановить образец, когда ассоциированный с ним образец предлагается ей в качестве подсказки. В этой сети два слоя элементов — по одному для каждого из образов пары — и оба слоя соединяются двухнаправленными связями (т.е. активность может передаваться по связям в обоих направлениях) (рис. 4.4).

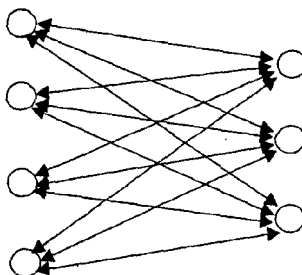


Рис. 4.4. Двухнаправленная ассоциативная память. Элементы слева представляют образцы размерности 4, а элементы справа — ассоциированные с ними образцы размерности 3

Здесь мы рассмотрим только дискретную биполярную сеть ВАРМ, но можно рассмотреть и непрерывные значения. Чтобы сохранить образец s и ассоциируемый с ним образец t , рассматривается прямое произведе-

ние, определяющее весовые значения. Процедура точно такая же, как и в сети Хопфилда, но теперь матрица уже не обязана быть квадратной, а диагональные элементы не обнуляются. Весовой матрицей для одной пары будет матрица

$$W = s^T t.$$

Чтобы сохранить несколько пар, все соответствующие произведения, определяющие весовые значения, складываются, точно так же, как это делается для сети Хопфилда.

Процедура нахождения в памяти элемента подобна соответствующей процедуре сети Хопфилда. В следующем описании i обозначает один слой элементов, а j — ассоциированный слой элементов.

- Устанавливаются значения активности элементов слоя i в соответствии со значениями, задаваемыми входным образцом.
- Распространяется активность на слой j . Комбинированный ввод элемента слоя j равен

$$net_j = \sum_i s_i w_{ij}.$$

- Вычисляется новое состояние для каждого элемента слоя j :

$$t_j = f(net_j).$$

- Распространяется активность на слой i . Комбинированный ввод элемента слоя i равен:

$$net_i = \sum_j s_j w_{ji}.$$

- Вычисляется новое состояние для каждого элемента слоя i :

$$s_i = f(net_i).$$

- Это двустороннее распространение сигналов активности повторяется до тех пор, пока не будет достигнуто устойчивое состояние. Активность для каждого слоя определяется относительно некоторой пороговой величины θ .

$$t_j = f(net_j) = \begin{cases} 1, & \text{если } net_j > \theta_j, \\ t_j, & \text{если } net_j = \theta_j, \\ -1, & \text{если } net_j < \theta_j, \end{cases}$$

$$s_i = f(net_i) = \begin{cases} 1, & \text{если } net_i > \theta_i, \\ s_i, & \text{если } net_i = \theta_i, \\ -1, & \text{если } net_i < \theta_i. \end{cases}$$

Все элементы сети сначала имеют нулевые значения активности. Обратите внимание на то, что распространение может начаться с любого уровня, так как и s может использоваться для вызова t , и, наоборот, — t может использоваться для вызова s .

Пример 4.4.

1. На рис. 4.5 показаны три образца (изображения цифр 1, 2 и 3). Эти три образца должны быть сохранены биполярной сетью ВМ. Ассоциируемыми с ними образцами являются трехбитовые двоичные числа (конвертированные в биполярную форму). Ассоциированные образцы представлены в табл. 4.2. Предложите весовые значения для этой сети.
2. Покажите, что каждая ассоциация может быть вызвана из памяти.

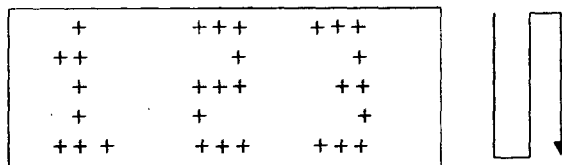


Рис. 4.5. Каждая цифра изображена на сетке 5×3 . Пробел кодируется значением -1 , а знак $'+'$ — значением $+1$. Цифры представляются линейным массивом значений, получаемым при движении по сетке сверху вниз и слева направо

Таблица 4.2. Цифры {1,2,3}, ассоциированные с биполярными образцами

Образец	Ассоциированный образец
1	-1 -1 1
2	-1 1 -1
3	-1 1 1

Решение 4.4.

1.

$$W = \begin{bmatrix} -1 \\ 1 \\ -1 \\ -1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ -1 \\ -1 \\ -1 \\ -1 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & -1 & 1 \end{bmatrix} + \begin{bmatrix} 1 \\ -1 \\ 1 \\ 1 \\ 1 \\ -1 \\ -1 \\ 1 \\ 1 \\ 1 \\ -1 \\ 1 \\ -1 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 1 & -1 \end{bmatrix} + \begin{bmatrix} 1 \\ -1 \\ -1 \\ -1 \\ 1 \\ 1 \\ -1 \\ 1 \\ 1 \\ -1 \\ -1 \\ -1 \\ -1 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 1 & 1 \end{bmatrix},$$

$$\therefore W = \begin{bmatrix} -1 & 3 & -1 \\ 1 & -3 & 1 \\ 1 & 1 & -3 \\ 1 & 1 & -3 \\ -3 & 1 & 1 \\ -3 & 1 & 1 \\ 1 & -3 & 1 \\ -3 & 1 & 1 \\ 1 & -3 & 1 \\ -3 & 1 & 1 \\ -1 & 3 & -1 \\ -1 & 3 & -1 \\ -1 & 3 & -1 \\ 1 & 1 & 1 \\ -3 & 1 & 1 \end{bmatrix}$$

2. Сначала в качестве входного рассмотрим образец, представляющий оцифрованное изображение цифры "2":

комбинированный ввод для слоя j

$$= [1 \ -1 \ 1 \ 1 \ 1 \ 1 \ -1 \ 1 \ -1 \ 1 \ 1 \ 1 \ -1 \ 1]$$

$$\times \begin{bmatrix} -1 & 3 & -1 \\ 1 & -3 & 1 \\ 1 & 1 & -3 \\ 1 & 1 & -3 \\ -3 & 1 & 1 \\ -3 & 1 & 1 \\ 1 & -3 & 1 \\ -3 & 1 & 1 \\ 1 & -3 & 1 \\ -3 & 1 & 1 \\ -1 & 3 & -1 \\ -1 & 3 & -1 \\ -1 & 3 & -1 \\ 1 & 1 & 1 \\ -3 & 1 & 1 \end{bmatrix}$$

$$= [-21 \ 27 \ -9].$$

С учетом порогового значения получаем $[-1 \ 1 \ -1]$, что и будет ассоциируемым образцом.

Образец $[-1 \ 1 \ -1]$ можно использовать в качестве входного:

комбинированный ввод для слоя j

$$= [-1 \ 1 \ -1] \times \begin{bmatrix} -1 & 3 & -1 \\ 1 & -3 & 1 \\ 1 & 1 & -3 \\ 1 & 1 & -3 \\ -3 & 1 & 1 \\ -3 & 1 & 1 \\ 1 & -3 & 1 \\ -3 & 1 & 1 \\ 1 & -3 & 1 \\ -3 & 1 & 1 \\ -1 & 3 & -1 \\ -1 & 3 & -1 \\ -1 & 3 & -1 \\ 1 & 1 & 1 \\ -3 & 1 & 1 \end{bmatrix}^T = \begin{bmatrix} 5 \\ -5 \\ 3 \\ 3 \\ 3 \\ 3 \\ -5 \\ 3 \\ -5 \\ 3 \\ 5 \\ 5 \\ 5 \\ -1 \\ 3 \end{bmatrix}^T.$$

Оригинальный соответствующий изображению образец воссоздается после прохождения полученного вектора через функцию активности.

Если процесс повторить для других векторов, будет видно, что вызываются все ассоциации.

4.4. Автоассоциативное обратное распространение ошибок

Стандартная сеть с прямой связью и обратным распространением ошибок, рассмотренная в главе 2, может быть обучена автоассоциативным методом выполнению задач типа сжатия изображений. Этот метод дает возможность сделать целевой образец таким же, как и учебный образец, так что сеть учится воспроизводить в выходном слое то, что подается ей на рассмотрение во входном слое. Базовая архитектура сети показана на рис. 4.6. Сеть обучается стандартным образом, но целью обучения является ассоциация каждого учебного образца с самим собой. После успешного окончания обучения сеть может работать как два механизма: первый слой весов может обеспечивать сжатие образца, а второй слой весов может восстанавливать полный образец из его сжатого представления. Конечно, в результате реконструкции могут быть и потери информации, поскольку сеть не будет в совершенстве создавать на выходном уровне все учебные экземпляры. Образец сжимается после его подачи на вход сети и распространения сигналов, по которым вычисляются значения активности скрытых элементов. Для сети с архитектурой типа

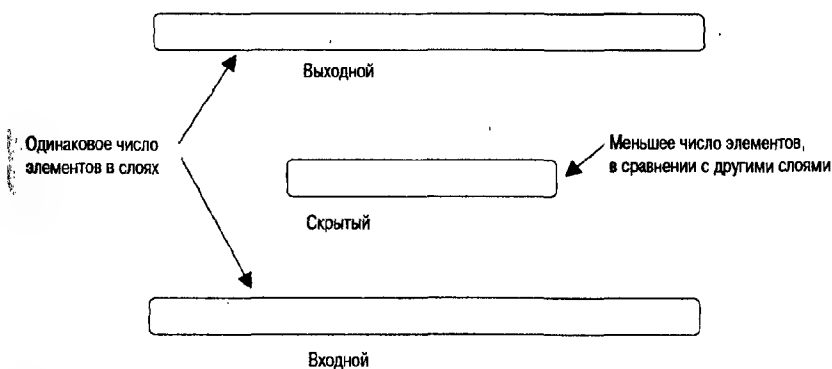


Рис. 4.6. Автоассоциативная сеть с прямой связью

20-10-20 (20 входных элементов, 10 скрытых элементов и 20 выходных элементов) отношением сжатия будет 2-1, поскольку для данного входного вектора сжатый вектор задается значениями активности скрытых элементов. Чтобы восстановить входной вектор, сжатый вектор предъявляется скрытому слою, и активность распространяется на выходной слой.

Иногда может оказаться выгодным обработать данные с помощью автоассоциативной сети перед тем, как рассмотреть их с помощью другой сетевой модели (и даже перед тем, как выполнить их классификацию с помощью другой сети с прямой связью). Известно, например, что автоассоциативная сеть с прямой связью, имеющая один скрытый слой, вычисляет по сути главные компоненты. Анализ главных компонент является основным методом статистической обработки данных и используется для того, чтобы удалить избыточность данных и провести кластеризацию. В результате анализа главных компонент происходит также декорреляция признаков векторов (признаки характеризуют положение элементов), что иногда оказывается полезным при обучении другой сети с прямой связью. Идея заключается в преобразовании учебных данных в некоторое сокращенное описание, где компонентами такого сокращенного описания выступают скрытые элементы.

Анализ главных компонент можно использовать для выполнения сжатия с потерями. Например, на рис. 4.7 показана некоторая совокупность двумерных данных. Через эту совокупность данных проведена новая пара осей, пересекающихся под прямым углом. Точки данных теперь могут быть выражены в координатах этих новых “главных осей”. Более длинная ось указывает направление наибольшего разброса или дисперсии данных. Если дисперсия в направлении более длинной главной оси составляет 85% разброса данных, а в направлении второй оси — 15%, то 85% дисперсии данных сохранится, даже если вторую главную координату из рассмотрения исключить. Другими словами, если данные описать только в терминах наибольшего главного компонента, то в основном информация о данных сохранится, но некоторая ее часть все же будет потеряна. Иногда в данных присутствует некоторая избыточность, как показано на рис. 4.8. Точки данных размещаются на прямой $x_2 = x_1$. Любая точка данных может быть выбрана в качестве единственной “главной координаты” без какой бы то ни было потери информации. Поэтому если данные выразить через такую главную координату, то каждый образец будет описан одним числом вместо двух. На первый взгляд, то, что образцы могут быть описаны одной координатой вместо двух (но только в случае, когда имеется избыточность, не забывайте!), может показаться странным. Ознакомьтесь тогда с табл. 4.3, в которой набор двумерных

точек описывается с помощью представления их значением одной координаты. Трансформация двумерных образцов в одномерные происходит следующим образом:

$$\begin{bmatrix} 1 & 1 \\ 3 & 3 \\ 4 & 4 \\ 8 & 8 \end{bmatrix} \begin{bmatrix} 1 \\ \frac{1}{\sqrt{2}} \\ 1 \\ \frac{1}{\sqrt{2}} \end{bmatrix} = [1.414 \quad 4.243 \quad 5.657 \quad 11.314].$$

Таблица 4.3. Примеры представления точек прямой $x_2 = x_1$ значениями одной координаты

x_1	x_2	Одно значение координаты
1	1	1.414
3	3	4.243
4	4	5.657
8	8	11.314

Обратное отображение для 1.414 задается с помощью следующего выражения:

$$[1.414] \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} = [1 \quad 1].$$

Точно так же могут быть восстановлены и другие координаты.



Рис. 4.7. Анализ главных компонент может использоваться для представления данных с помощью нового набора координатных осей

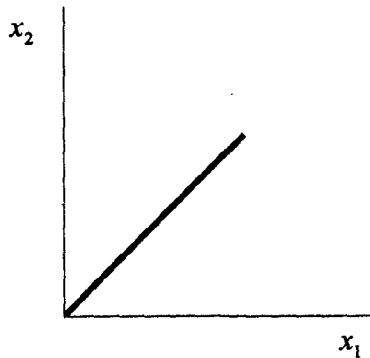


Рис. 4.8. Эта прямая состоит из точек, описываемых двумя координатными значениями, но может быть представлена и с помощью одной координаты (т.е. в одномерном виде)

Более высокого показателя сжатия можно иногда достичь с помощью дополнительных скрытых слоев. Например, сеть с архитектурой 40-20-8-20-40 может обеспечить отношение сжатия 40 к 8, если, конечно, ее обучение пройдет успешно.

4.5. Резюме

Ассоциативная сеть действует подобно памяти.

- Автоассоциация связывает образец с ним самим. Например, автоассоциативная память может использоваться для того, чтобы восстановить истинную версию сохраненного образца по искаженной версии этого образца.
- Гетероассоциация представляет собой связывание двух разных образцов. Один образец может использоваться в качестве подсказки, по которой из памяти извлекается другой образец.
- Когда автоассоциативное обучение происходит путем пропуска сигналов через узкий канал типа скрытого слоя автоассоциативной сети с прямой связью, происходит в некотором смысле сжатие данных. Сжатие может оказаться полезной формой предварительной обработки данных перед использованием их для обучения сети с другой архитектурой.

- Сеть Хопфилда может использоваться для автоассоциации, а двунаправленная ассоциативная память — для гетероассоциации.
- Весовые значения дискретной сети Хопфилда и двунаправленной ассоциативной памяти могут быть найдены с помощью простых матричных вычислений, поэтому такие сети не требуют длительного обучения.
- От числа элементов в ассоциативной сети зависит число образцов, которые могут быть сохранены.

4.6. Дополнительная литература

Подробное обсуждение сети Хопфилда и автоассоциативной сети с обратным распространением ошибок имеется в книге [Haykin, 1994].

4.7. Упражнения

1. Определите весовые значения сети Хопфилда, соответствующие сохранению образца $[1 \ 1 \ 1 \ -1]$.
2. Для сети из упражнения 1 рассмотрите подсказку в виде входного образца $[1 \ 1 \ 1 \ -1]$ и проверьте, стабилизируется ли сеть на сохраненном образце.
3. Повторите вычисления упражнения 2 для образца $[-1 \ -1 \ 1 \ -1]$.
4. Сколько образцов можно вызвать из сети Хопфилда, если каждый сохраненный вектор имеет 10 элементов?
5. (а) Определите весовые значения сети Хопфилда для сохранения следующих образцов:

$$[-1 \ 1 \ -1 \ -1 \ 1 \ -1 \ -1 \ -1],$$

$$[-1 \ -1 \ 1 \ -1 \ -1 \ -1 \ 1 \ -1].$$

- (б) Проверьте устойчивость сети при предоставлении ей на вход в качестве подсказок сохраненных образцов.
- (в) Проверьте устойчивость сети при предоставлении ей на вход в качестве подсказок следующих образцов:

$$[-1 \ 1 \ -1 \ -1 \ 1 \ -1 \ -1 \ 1],$$

$$[-1 \ 1 \ 1 \ -1 \ 1 \ -1 \ -1 \ -1].$$

6. Определите весовые значения сети Хопфилда для сохранения следующих образцов:

$[-1 \ 1 \ -1 \ -1 \ 1 \ -1 \ -1 \ -1],$

$[-1 \ -1 \ 1 \ -1 \ -1 \ -1 \ 1 \ -1],$

$[1 \ -1 \ -1 \ 1 \ -1 \ 1 \ -1 \ -1].$

Проверьте устойчивость сети при предоставлении ей на вход в качестве подсказки первого сохраненного образца.

7. Определите сеть ВАН для примера 4.4, но в качестве ассоциированных образцов для изображений цифр возьмите следующие векторы.

Образец	Ассоциированный образец
1	-1 1
2	1 -1
3	1 1

Сравните полученные результаты с результатами, представленными в описании примера 4.4.

Глава 5

Рекуррентные сети

Задача. Описание сетей с рекуррентными связями.

Цели. Вы должны понять:

для чего предназначены рекуррентные сети и в чем заключаются их преимущества по сравнению с другими сетями при решении задач определенного типа;
как для конкретной задачи построить сеть, не являющуюся рекуррентной, но эквивалентную рекуррентной по получаемым на выходе результатам;
как модифицировать алгоритм обратного распространения ошибок для его реализации в простой рекуррентной сети.

Требования. Знакомство с материалом глав 1 и 2.

5.1. Введение

В этой главе дается краткий обзор некоторых структур, предназначенных для обработки последовательностей. *Последовательность* — это цепочка образцов, имеющих отношение к одному и тому же объекту. Например, последовательность может состоять из букв, образующих слово, или из слов, образующих предложение. Иметь дело с последовательностями не так уж просто, поскольку они могут иметь разную длину. Например, число входящих в предложение слов может меняться в довольно широких пределах. Если учебным образцам позволить иметь любое число элементов, то сколько тогда должно быть входных элементов в сети? Одним из способов обработки предложений с помощью фиксированного числа входных элементов является использование скользящего окна, как показано на рис. 5.1. Недостатком этого подхода является то, что последовательность необходимо делить на сегменты, в результате чего зависимость между находящимися далеко друг от друга словами теряется. Чтобы сразу обрабатывать все предложение, потребуется нейронная сеть с архитектурой, допускающей обработку самого длинного из ожидаемых предложений. Эта проблема решается путем использования сети с рекуррентными связями.

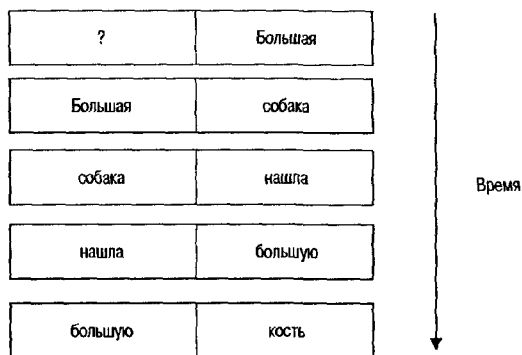


Рис. 5.1. Предложение "Большая собака нашла большую кость" подается на вход сети. Входной слой сети имеет два поля элементов, чтобы рассматривать в каждый момент времени два слова. До того как сети встретится последнее слов "кость", ей придется четыре раза обработать по два слова (текущее плюс предыдущее)

5.2. Обратное распространение во времени

Сеть с обратным распространением ошибок, описанная в главе 2, является сетью с прямой связью, что означает ограничение направления связей: все связи должны идти в одном направлении — от входного слоя к выходному. Но сеть с обратным распространением ошибок не обязательно должна быть сетью с прямой связью, она может иметь и рекуррентные связи, когда элемент посылает сигналы себе или другим элементам того же слоя или более низких слоев. Фактически допустимым является любой вариант обратной связи. Обратную связь, задаваемую рекуррентными связями, не следует путать с обратной связью ошибок при корректировке весов. Обратная связь ошибок является частью процедуры корректировки весов, тогда как рекуррентная связь подает обратно значение активности, которое будет влиять на выход сети в течение следующих итераций.

Алгоритм работы рекуррентной сети с обратным распространением ошибок представляет собой непосредственную модификацию соответствующего алгоритма работы сети с прямой связью. Эта модификация опирается на утверждение, что для любой рекуррентной сети существует сеть с прямой связью, имеющая идентичное поведение. На рис. 5.2 пока-

зана сеть со связями, по которым сигналы возвращаются обратно от выходного слоя к входному, а рис. 5.3 иллюстрирует работу этой сети в течение двух шагов во времени. Обратите внимание на то, что дополнительные слои с направленными вперед взвешенными связями дублируют направленные вперед связи рекуррентного варианта сети. Другими словами, чтобы имитировать рекуррентную сеть, она дублируется для двух шагов выполнения во времени.

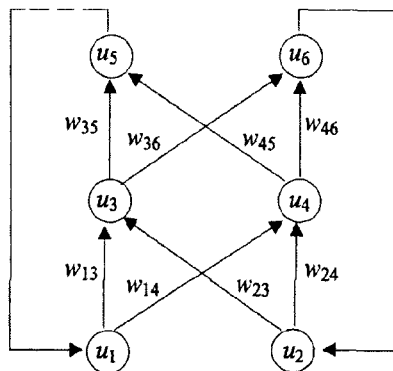


Рис. 5.2. Рекуррентная сеть с обратным распространением ошибок

В ходе обучения на вход сети подается образец и выполняется прямой проход. Для каждого шага времени рассматривается своя копия сети и для каждого шага времени вычисляются ошибки точно так же, как в стандартной сети с обратным распространением ошибок. Изменения весовых значений, вычисленные для каждого экземпляра сети, перед корректировкой суммируются. Наборы весовых значений всех экземпляров сети (или шагов времени) должны оставаться одинаковыми.

В [Rumelhart *et al.*, 1986a] представлен ряд экспериментов с рекуррентными связями. В одном из этих экспериментов происходит обучение сети выполнению задачи дополнения последовательности. Последовательность состоит из шести символов, причем первые два символа должны быть буквами, выбираемыми из множества {A, B, C, D, E}, а остальные — целыми числами, выбираемыми из множества {1, 2, 3}. Первые две буквы определяют оставшуюся часть последовательности. Например, если “A” определяет последовательность “12”, а “C” — последовательность “13”, то вся последовательность, начинающаяся буквами “AC”, должна иметь вид “AC1213”, а вся последовательность, начинающаяся буквами “AA”, — вид “AA1212”.

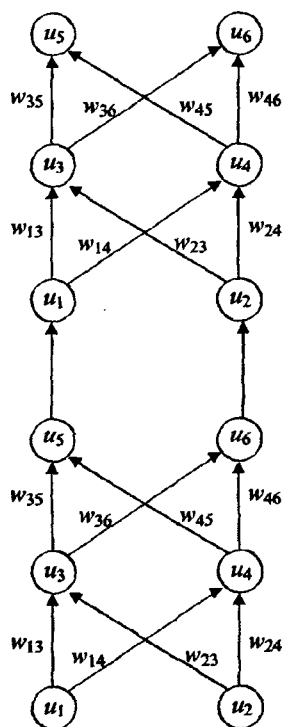


Рис. 5.3. Развернутая версия сети, показанной на рис. 5.2

Сеть содержала пять входных элементов (по одному на каждую букву), три выходных элемента (по одному на каждую цифру) и 30 скрытых элементов. Все скрытые элементы были связаны сами с собой и всеми остальными скрытыми элементами, и все выходные элементы были связаны сами с собой и всеми остальными выходными элементами. Обучение осуществлялось в результате выполнения следующих шагов.

Шаг 1. Для входного элемента, соответствующего первой букве, устанавливается значение активности, равное 1, а все остальные входные элементы переводятся в выключенное состояние.

Шаг 2. Для входного элемента, соответствующего второй букве, устанавливается значение активности, равное 1, а все остальные входные элементы переводятся в выключенное состояние.

Все входные элементы переводятся в выключенное состояние.

Шаг 3. В качестве целевого выходного значения рассматривается третий символ (т.е. первое число).

Шаг 4. В качестве целевого выходного значения рассматривается четвертый символ (т.е. второе число).

Шаг 5. В качестве целевого выходного значения рассматривается пятый символ (т.е. третье число).

Шаг 6. В качестве целевого выходного значения рассматривается шестой символ (т.е. четвертое число).

Все скрытые и выходные элементы в начале получали значения активности, равные 0.2. Сеть обучалась на 20 последовательностях. Затем на пяти тестовых примерах было продемонстрировано, что сеть действительно могла завершить последовательности по заданным двум первым символам. Другой эксперимент демонстрировал, что последовательности успешно могли быть дополнены даже в случаях, когда имели место задержки при предъявлении сети первой и второй букв (например, “С, *задержка*, А; *задержка*, 1312”).

Пример 5.1.

Рассматривается полносвязная сеть с прямой связью и архитектурой 3-2-2, имеющая рекуррентные связи: каждый скрытый элемент связан с самим собой и со всеми другими скрытыми элементами, и каждый выходной элемент связан с самим собой и со всеми другими выходными элементами. Изобразите схему такой сети и схему эквивалентной ей сети без рекуррентных связей для одного шага времени. Один шаг времени в данном примере означает обработку входного образца двумя скрытыми элементами и двумя выходными элементами.

Решение 5.1.

Соответствующие схемы показаны на рис. 5.4 и 5.5.

Модели нейронных сетей, не имеющие ограничений на число и тип рекуррентных связей, могут при первом рассмотрении показаться слишком сложными. Сеть, обсуждаемая в следующем разделе, легче для понимания, поскольку оказывается простой модификацией стандартной сети с обратным распространением ошибок. Эта сеть, называемая простой рекуррентной сетью, несмотря на ее примитивную архитектуру, используется очень широко.

Модели нейронных сетей, не имеющие ограничений на число и тип рекуррентных связей, могут при первом рассмотрении показаться слишком сложными. Сеть, обсуждаемая в следующем разделе, легче для понимания, поскольку оказывается простой модификацией стандартной сети с обратным распространением ошибок. Эта сеть, называемая простой рекуррентной сетью, несмотря на ее примитивную архитектуру, используется очень широко.

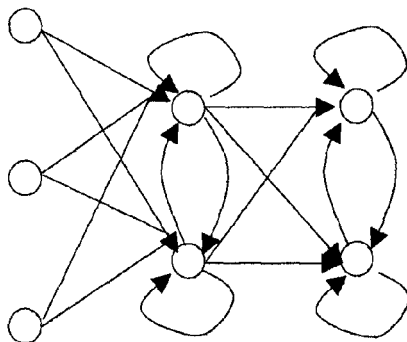


Рис. 5.4. Сеть с полным набором рекуррентных связей в скрытом и выходном слоях

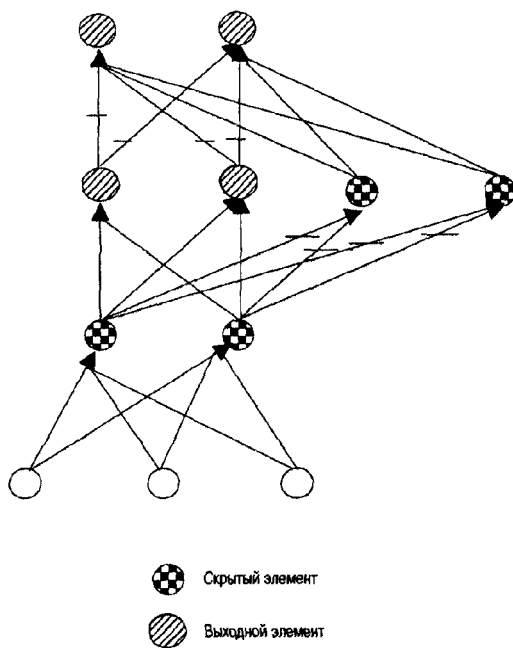


Рис. 5.5. Перечеркнутые связи соответствуют рекуррентным связям на рис. 5.4. Соответствующие два набора весовых значений связей между элементами скрытого и выходного слоев одинаковы

5.3. Простая рекуррентная сеть

В своем докладе [Jordan, 1989] Джордан представил одну из наиболее ранних форм рекуррентной сети. Сеть Джордана имеет связи, идущие обратно от выходного к входному слою, и некоторые элементы входного слоя имеют связи, возвращающиеся обратно к этим элементам. Архитектура сети показана на рис. 5.6. Сеть способна обучиться выполнению задач, зависящих от последовательности состояний. Такая сеть может быть обучена с помощью алгоритма обратного распространения ошибок.

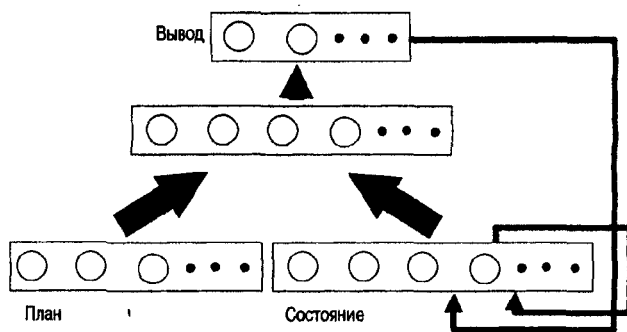


Рис. 5.6. Сеть Джордана

Из-за рекуррентных связей в сети Джордана поведение сети формируется введенными ранее входными данными. Сеть обладает некоторыми свойствами краткосрочной памяти. Еще одной очень популярной рекуррентной сетью с подобными свойствами краткосрочной памяти является так называемая *простая рекуррентная сеть* (сеть SRN — Simple Recurrent Network).

В [Elman, 1990] было продемонстрировано, что сеть SRN может предсказать следующий элемент последовательности по текущему и предшествующему вводу. В сети SRN скрытые элементы имеют связи, направленные обратно во входной слой, как показано на рис. 5.7. Развернутая версия сети SRN для трех шагов показана на рис. 5.8. Скрытые элементы составляют внутреннее редуцированное представление данных, предшествующих текущему вводу. Это редуцированное представление образует контекст, оказывающийся для определенных задач существенным. Например, третий элемент в последовательностях {110, 000, 101, 011} не может быть однозначно предсказан по второму или первому элементам, рассматриваемым независимо. Но третий элемент зависит от совокупности первого и второго элементов, рассматриваемых вместе. Первые два

элементы этих последовательностей соответствуют двоичным данным ввода для проблемы XOR, и Элман (Elman) продемонстрировал, что сеть SRN может быть обучена решению проблемы XOR.

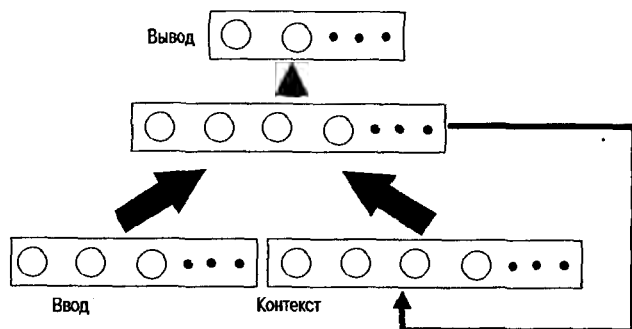


Рис. 5.7. Простая рекуррентная сеть. Число контекстных элементов равно числу скрытых элементов

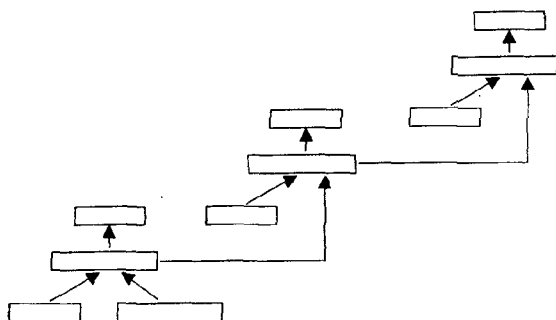


Рис. 5.8. Развернутая версия сети SRN, представленной на рис. 5.7

В случае проблемы XOR сеть SRN обучается на длинной последовательности битов. В этой последовательности первый и второй биты связываются операцией XOR, чтобы получить третий бит, четвертый и пятый биты связываются операцией XOR, чтобы получить шестой, и т.д. Сеть должна научиться генерировать следующий элемент последовательности. Вот пример ввода и вывода сети:

ввод: 1 0 1 0 0 0 1 1 1 1 0 1 0 1 ...,

вывод: 0 1 0 0 0 0 1 1 1 1 0 1 0 1 ?

В эксперименте сеть SRN, состоящая из одного входного элемента, двух контекстных элементов (следовательно, двух скрытых элементов) и одного выходного элемента, обучалась на последовательности 3000 битов. Результат обучения показал, что сеть выучила временную структуру последовательности, так как для каждого третьего бита квадрат ошибки должен оказаться малым. Существенное уменьшение величины ошибки вполне адекватно характеризует временную структуру последовательности. Например, сеть всегда может пытаться связать операцией XOR предыдущие два бита, но правильность такого предсказания будет гарантировать только каждый третий бит. Рассмотрим последовательность 101110. Третий и шестой биты являются результатами применения операции XOR к предыдущим двум битам, поэтому ошибка предсказания оказывается малой для третьего и шестого битов.

5.3.1. Применение сети SRN

Обучение простым правилам грамматики

В работе [Cleeremans, 1993] представлено несколько интересных экспериментов с сетями SRN. Одной из задач было обучение сети SRN грамматике Ребера (Reber), показанной на рис. 5.9. Эта грамматика схематически представляется в виде конечного автомата (автомата конечных состояний). С помощью такого автомата может генерироваться строка символов в соответствии с правилами указанной грамматики. Можно также проверить, будет ли такая строка грамматически правильной.

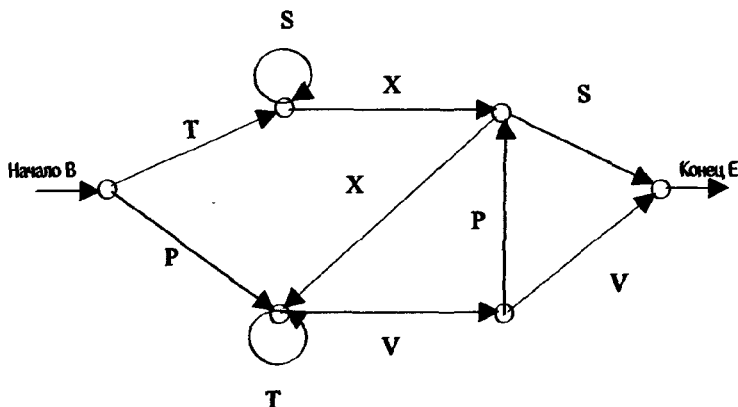


Рис. 5.9. Конечный автомат для грамматики Ребера

Чтобы сгенерировать строку, выберите сначала узел В, затем — дугу, идущую к другому узлу. Каждый раз, когда дуга проходит через узел, записывается соответствующая этому узлу метка (символ). Когда будет достигнута конечная вершина Е — из нее можно перейти только к выходу — строка обрывается. Чтобы некоторая строка оказалась грамматически правильной, необходимо породить эту строку в соответствии с дугами, заданными в автомате. Ниже приведены примеры строк.

В Р V Р X Т V Р X V V E,
 В Р V Р X V V E,
 В Т X S E,
 В Р V V E,
 В Т S S X S E,
 В Р T T T V V E.

Мы повторим один из экспериментов Клеерманса (Cleeremans) с небольшими модификациями. Напомним, что целью является обучение сети SRN указанной грамматики. Используем шесть входных и шесть выходных элементов, где каждый элемент представляет один символ. Элементы входного и выходного слоев размещены так, как показано на рис. 5.10. С помощью специальной программы случайным образом генерировались 20000 строк в соответствии с грамматикой Ребера (Клеерманс использовал 60000 строк). В ходе обучения во входной слой подавался один символ строки, а следующий символ рассматривался в качестве целевого выхода. Для всех последовательностей контекстные элементы начинали обучение со значения 0.5. После прохождения первого символа через сеть, на вход сети подавался второй символ, а третий выполнял роль целевого выходного значения. Перед рассмотрением очередного ввода значения активности скрытых элементов копировались на контекстные элементы. Процесс повторялся для каждого из символов последовательности, а вся процедура в целом была выполнена для каждого из 20000 образцов. Норма обучения и коэффициент инерции были выбраны равными 0.1, и при обучении сети были выполнены три прохода по всем образцам. Весовые значения обновлялись после рассмотрения каждого символа.

Результаты обучения показывают, что сеть моделирует вероятность следования одного символа за другим. Например, в каждом узле грамматики Ребера одна из двух дуг может быть выбрана с равной вероятностью (учебные строки генерировались так, что любой путь выбирался с одинаковой вероятностью). Интересная особенность данной грамматики заключается в том, что каждый символ представляется двумя дугами, а следующее допустимое состояние зависит от того, по какой дуге осуществляется

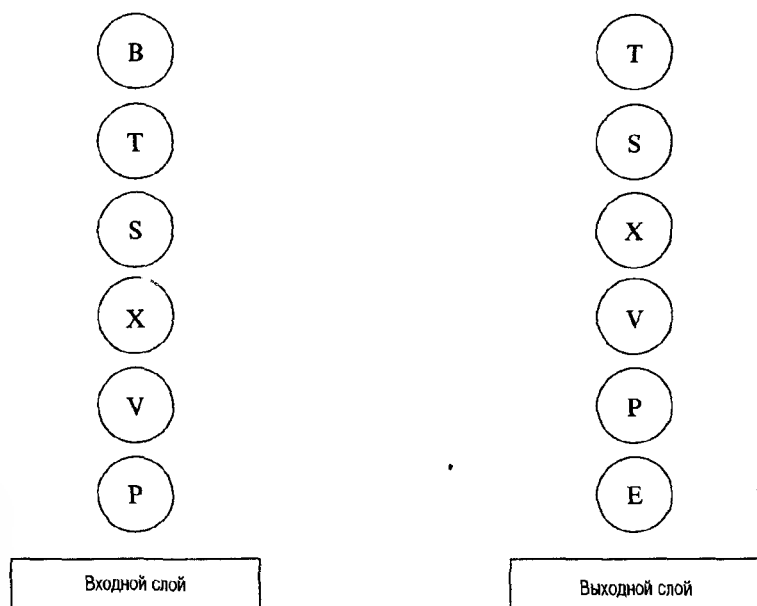
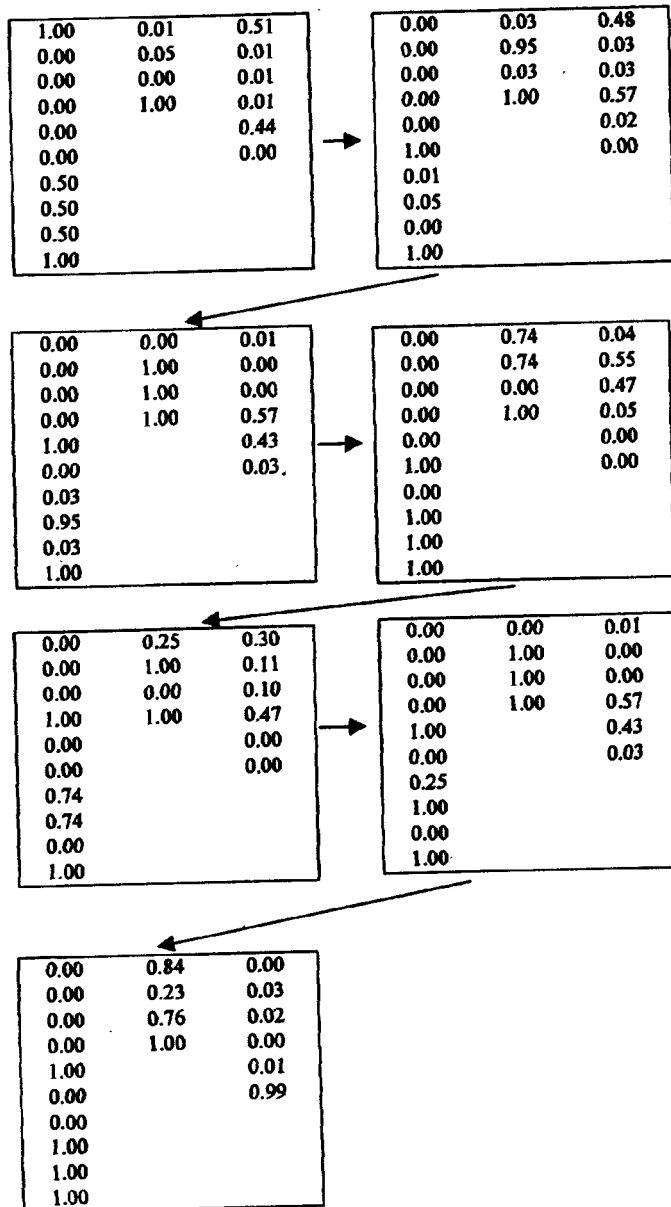


Рис. 5.10. Элементы входного и выходного слоев

переход. Например, за символом P, следующим за начальным состоянием, может последовать T или V, но за другим P в грамматике могут следовать только S или X. Таким образом, за символом P может следовать один из четырех различных символов, но в любой точке последовательности будут допустимыми только два из них.

На рис. 5.11 показаны последовательные состояния всех элементов сети при обработке последовательности B P V P X V V. Для каждого состояния высокую активность будут иметь два выходных элемента, поскольку для следующего состояния имеется два возможных варианта. Начальное состояние B активизирует T и P в выходном слое, поскольку именно они с равной вероятностью являются допустимыми последователями. Первый символ P определяет возможными последователями T и V, но, как и ожидалось, второй символ P в качестве последователей указывает X и S.

Для тестирования своей обученной сети SRN Клеерманс использовал 20000 случайным образом сгенерированных в соответствии с грамматикой строк, чтобы проверить, признает ли их сеть грамматически правиль-



ными. Сеть признала правильными все 20000 строк. Затем сеть была протестирована на множестве из 130000 строк, которые были сгенерированы случайным образом, но не обязательно по правилам грамматики. Точнее говоря, строка строилась с помощью последовательного случайного выбора одного из пяти символов (T, S, X, V, P). Из 130000 строк 260 оказались грамматически правильными, а остальные (99.8%) были грамматически неправильными. Сеть выполняла свою работу в совершенстве, отвергая недопустимые с точки зрения грамматики строки и признавая грамматически правильные. Сеть также правильно обрабатывала очень длинные строки (порядка 100 символов).

Обучение сложению чисел

Теперь давайте рассмотрим использование сети SRN для сложения чисел. Для простоты мы ограничимся сложением только целых значений. Принимая восьмибитовое представление, число 11 будет двоичным 00001011, а число 9 — двоичным 00001001. Сложение в двоичном виде выполняется так:

$$\begin{array}{r} 00001011 \\ 00001001 \\ \hline 00010100 \end{array}$$

Сложение двоичных чисел подобно сложению десятичных. Значения разрядов складываются по столбцам по очереди, справа налево. Если сумма в столбце оказывается больше 1, в результат записывается 0, а 1 переносится в следующий столбец. В каждый момент обрабатываются только две цифры, и их комбинации должны быть следующими:

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix},$$

где в нижней строке приводится сумма двух цифр из верхних строк.

Рис. 5.11. Значения активности элементов сети при обработке BPVPXVV. Первые шесть элементов входного слоя представляют символы, а следующие три являются контекстными элементами. Последними элементами входного и скрытого слоев являются элементы смещения, поэтому установленные для них значения оказываются всегда равными 1

Результат из нижней строки может быть интерпретирован как результат операции XOR (операция исключающего “ИЛИ”) для двух суммируемых цифр. Перенос единицы происходит тогда, когда обе суммируемые цифры равны 1. Результатом операции AND (операции “И”) для двух этих битов будет 1. Таким образом, сумма, которая должна быть записана, может быть получена с помощью логического элемента XOR, а перенос — с помощью логического элемента AND, как показано на рис. 5.12. Эта схема называется *полусумматором*, и, дополнив эту схему еще несколькими элементами логики, можно реализовать двоичное сложение в полной мере. Мы уже знаем, что нейронная сеть может быть обучена осуществлению операции XOR, а схему AND осуществить еще проще. Поэтому можно использовать модульную нейронную сеть, выполняющую сложение в двоичной системе счисления. Так как основу вычислений на уровне аппаратных средств образуют именно логические элементы, можно на основе элементарных схем нейронной сети создать подобные аппаратные средства, имитирующие обычный компьютер. Однако при моделировании обычных аппаратных вычислительных средств с помощью нейронных сетей имеется особенность, состоящая в том, что придется искать решение, полученное в результате обучения.

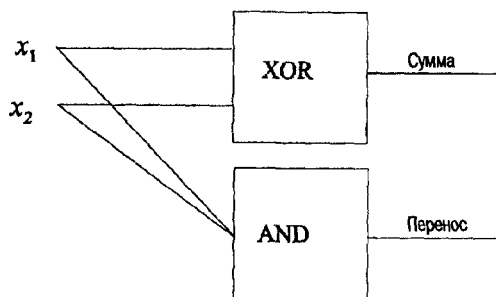


Рис. 5.12. Логическая схема, называемая полусумматором

Складывая большие числа (если мы не обладаем исключительными способностями), нам приходится использовать карандаш и бумагу, чтобы записывать результаты промежуточных стадий вычисления. Мы начинаем суммирование с крайнего столбца справа, записываем сумму первых двух разрядов и, если необходимо, отмечаем на бумаге единицу переноса (рис. 5.13).

$$\begin{array}{r}
 1 \\
 1\ 2\ 5 \\
 3\ 5\ 7 \\
 \hline
 4\ 8\ 2
 \end{array}$$

Рис. 5.13. Сложение двух чисел с помощью карандаша и бумаги

В статье [Noelle, Cottrell, 1995] описана реализация сумматора на основе нейронной сети, в котором нет явного учета информации о переносе из одного столбца в другой. Вместо этого, информация о переносе отражается состоянием сети. Авторы использовали два типа нейронных сетей: простую рекуррентную сеть (сеть SRN) и сеть Джордана. Мы рассмотрим здесь сеть SRN, но аналогичная процедура использовалась и для сети Джордана. Базовая архитектура сети представлена на рис. 5.14.

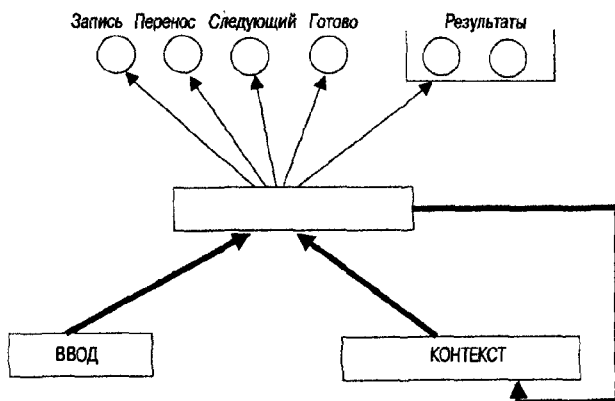


Рис. 5.14. Сеть SRN для сложения цифр

Элемент “запись” указывает, когда результат должен быть записан, элемент “перенос” — когда имеется информация о переносе, элемент “следующий” — когда должна быть представлена пара значений следующего разряда, а элемент “готово” — когда суммирование завершено. Шаги, выполняемые при сложении чисел 865 и 327, представлены в табл. 5.1. Во всех экспериментах использовались вычисления по основанию 4 (т.е. бит переноса возникал тогда, когда сумма двух цифр оказывалась большей или равной 4). Каждая цифра представлялась двумя элементами.

Таблица 5.1. Последовательность шагов при сложении 865 и 327 в сети SRN (рис. 5.14)

Шаг	1-я цифра	2-я цифра	Запись	Перенос	Следующий	Готово	Результат
1	7	5	✓				2
2	7	5		✓			
3	7	5			✓		
4	2	6	✓				9
5	2	6			✓		
6	3	8	✓				1
7	3	8		✓			
8	3	8			✓		
9	0	0	✓				1
10	0	0				✓	

Учебные данные состояли из чисел, имеющих не более трех разрядов. Начальное обучение на малых случайных подмножествах полного учебного набора данных дало слабые свойства обобщения: сеть просто запомнила все учебные данные. Тогда был использован метод *комбинированного обучения на подмножествах* (так его назвали авторы), при котором сначала обучение проходит на случайном малом подмножестве учебных данных, затем размеры этого подмножества удваиваются, пока не оказывается представленным весь набор учебных данных. Результаты выглядели впечатляюще. Сеть могла обобщить результаты обучения на все учебные пары уже после рассмотрения только 8% всего набора данных. Больше того, после в некотором смысле “рафинированного” обучения (заключающегося в рассмотрении небольшого подмножества трудных учебных примеров), сеть могла использовать полученные знания для решения задач, в которых использовались даже 14-разрядные числа.

Ноэль (Noelle) и Коттрелл (Cottrell) выяснили, чем отличается сеть Джордана от сети SRN. Сеть Джордана не могла выполнить задачу при внесении небольших изменений в операции вывода, когда информация уведомления о переносе задерживалась до завершения записи суммы следующего столбца. И сеть Джордана, и сеть SRN являются сетями с рекуррентной архитектурой, но сеть SRN непрерывно обновляет состояния, получаемые как внутренние представления.

Анализируя состояния скрытых элементов, Ноэль и Коттрелл показали, что сеть SRN работала подобно конечному автомату (т.е. рассмотренной вы-

ше структуре, моделирующей вычислительные устройства для представления грамматики). Известно, что с помощью сети SRN можно реализовать любой конечный автомат (см. [Kremer, 1995]). В принципе сеть SRN может оказаться полезной для моделирования многих типов вычислительных машин. Применение сети SRN для решения ряда других задач рассматривается в главе 8.

5.4. Резюме

Сети, рассмотренные в этой главе, используют простые модификации алгоритма обратного распространения ошибок. Эти модификации дают возможность использовать рекуррентные связи.

- Для любой рекуррентной сети имеется сеть с прямой связью, обладающая идентичным поведением.
- Рекуррентные сети используются для обработки образцов, имеющих переменную длину. Такие образцы переменной длины рассматриваются как последовательности: последовательность разбивается на отрезки данных, и каждый отрезок подается на рассмотрение сети на разных шагах.
- Простые рекуррентные сети могут демонстрировать способность предсказывать следующие отрезки последовательности данных по ранее введенным данным.

5.5. Дополнительная литература

Книга [Cleeremans, 1993], посвященная безусловному обучению, содержит ряд интересных примеров сетей SRN и впечатляющие результаты их применения. Другие примеры использования рекуррентных сетей будут представлены в главе 8.

5.6. Упражнения

1. Сеть SRN имеет пять входных элементов, восемь контекстных элементов и четыре выходных элемента. Сколько весовых значений имеет эта сеть?
2. Сеть SRN была обучена на множестве строк переменной длины, состоящих из символов. Символы всегда размещались по алфавиту. Например:

B C D E конец,
B C D E F G конец,
E F G H конец.

Сеть обучалась предсказывать следующий символ в последовательности. Не касаясь числа элементов, для сети SRN, выполняющей обработку последовательности "E F G H конец", постройте схему эквивалентной сети с прямой связью. Укажите ввод и целевой вывод на каждой стадии.

3. Сколько входных элементов вы использовали для выполнения упражнения 2? Обоснуйте свой выбор.
4. Сеть SRN обучается на последовательностях, состоящих из восьми символов и символа NIL: {B, R, A, E, T, N, S, I, NIL}. Порядок, в котором появляются эти символы в последовательности, не имеет значения, но каждая последовательность должна содержать по крайней мере одну гласную и одну согласную и заканчиваться символом NIL. Символы последовательности подаются на рассмотрение сети SRN в порядке слева направо, но целевым выходом всегда является полная последовательность: сеть обучается предсказывать полную последовательность по уже предъявленным ей первым символам.
 - (а) Предложите несколько учебных экземпляров.
 - (б) Предложите архитектуру сети SRN и объясните, почему вы выбрали именно такую архитектуру.

Глава 6

Другие модели сетей и практические вопросы

Задача. Описание других моделей сетей и обсуждение некоторых практических вопросов.

Цели. Вы должны понять:

что такое метод модельной “закалки” и как он используется в нейронных сетях;
принципы построения вероятностной нейронной сети, чтобы иметь возможность реализовать такую сеть в программе электронных таблиц;
что такое “машина Больцмана”;
какие проблемы возникают при использовании нейронных сетей.

Требования. Знакомство с материалом глав 1–5. Знание элементарных основ статистики.

6.1. Введение

В предыдущих главах были рассмотрены модели, лежащие в основе технологии использования нейронных сетей. Целью данной главы является рассмотрение некоторых других моделей и некоторых практических вопросов, возникающих при использовании нейронных сетей. Глава начинается с раздела о моделях сетей, в которых применяются статистические методы. Основным объектом рассмотрения будет вероятностная сеть, а метод модельной “закалки” и машина Больцмана будут рассмотрены только на самом начальном уровне, чтобы просто информировать читателя о существовании таких объектов. Этот раздел завершается примером модульной архитектуры, в основу которой положены модели,

представленные в предыдущих главах. Мы стремились показать, как можно создавать новые модели, и дать читателю понять, что иногда решения проблем, кажущихся на первый взгляд практически неразрешимыми, все же существуют.

6.2. Сети, использующие статистический подход

До сих пор все рассмотренные нами модели нейронных сетей использовали *детерминированные* алгоритмы обучения. Слово “детерминированные” здесь означает, что при необходимости обновления весовых значений или значения активности элемента соответствующие величины изменений могут быть определены в результате непосредственного прямого вычисления. Даже в сети Хопфилда, где обновляемый элемент выбирается случайно, сами изменения являются детерминированными, поскольку если элемент уже выбран, новое значение его активности может быть вычислено вполне однозначно. В противоположность этому, для *стохастического* алгоритма обучения в любой момент времени мы остаемся в неведении относительно того, как изменится состояние сети (“состояние” в данном случае означает набор текущих весовых значений и активность элемента). Другими словами, по информации о текущем состоянии предсказать следующее состояние оказывается невозможным. Например, при использовании стохастического алгоритма обновления состояния элемента в сети Хопфилда для принятия решения о том, должен данный элемент перейти в новое вычисленное состояние или нет, используется функция распределения вероятностей. Поэтому даже когда элемент выбран для обновления, непосредственного обновления его состояния может и не произойти. Но, хотя мы и не можем сказать, как стохастическая сеть будет реагировать в каждый конкретный момент времени, мы в общем на самом деле знаем, как будет вести себя сеть по прошествии достаточно долгого периода времени.

6.2.1. Метод модельной “закалки”

Метод модельной “закалки” является методом, часто используемым при решении проблем оптимизации. Решение рассматриваемой проблемы представляется в виде решения задачи минимизации некоторой функции стоимости. Функция стоимости определяется в терминах глобальной энергии сети. Например, определенная стоимость соответствует каждому маршруту, который принимает решение пройти продавец, чтобы посетить своих клиентов в разных городах. В данном случае стоимость

может быть выражена в терминах расстояния, которое предстоит преодолеть, и тогда функция стоимости возвратит длину выбранного маршрута. Маршруты с меньшим расстоянием будут иметь меньшую стоимость. Самым оптимальным маршрутом в данном случае будет маршрут наименьшей длины. Иногда невозможно найти оптимальное решение, тогда следует искать разумно оптимальное решение.

Метод модельной “закалки” базируется на аналогии с процессом закалки металла. Когда металл подвергается закалке, он нагревается почти до точки плавления, а потом медленно охлаждается до комнатной температуры. Процесс закалки делает металл более гибким, так что становится возможным придать ему нужную форму без изломов. Когда металл нагрет до высокой температуры, атомы двигаются хаотически. Если металл быстро охладить, атомы застынут в случайных положениях. Если же металл, напротив, охлаждать медленно, атомы будут стремиться выстроиться регулярным образом. Поэтому основой всего процесса является управление графиком снижения температуры.

Аналогию с функцией оптимизации иллюстрируют рис. 6.1 и 6.2. На рис. 6.1 показана кривая, заданная некоторой гипотетической функцией одной переменной. Эта функция не является гладкой и имеет несколько (локальных) минимумов. Процедура для нахождения глобального минимума является очень простой. Сначала выбирается диапазон, в котором будет оцениваться функция. Значения функции оцениваются в некотором числе случайно выбираемых точек, при этом запоминается наименьшее из них. Диапазон затем уменьшается, и значения функции оцениваются снова в некотором числе случайно выбираемых точек. Снова сохраняется наименьшее из этих значений. Процесс повторяется снова и снова с постепенным уменьшением диапазона.

Выбор диапазона и его сокращение аналогичен установке изначально высокой температуры и постепенному уменьшению этой температуры на последующих стадиях. Сначала точка может находиться везде без ограничений, “прыгая” от одного конца кривой к другому, поскольку диапазон изначально выбирается большим. Опять же, это напоминает хаотическое движение атомов металла при высокой температуре.

Еще одним применением метода модельной “закалки” является обход локальных минимумов. Детерминированные алгоритмы типа алгоритма обратного распространения, использующие метод градиентного спуска, часто приводят к локальным минимумам. Попав в такой минимум, сеть не может больше двигаться по поверхности ошибок к более оптимальному решению. Алгоритмы, подобные градиентному спуску, всегда пытаются продвинуться вниз, чтобы уменьшить величину полной ошибки, но

представленные в предыдущих главах. Мы стремились показать, как можно создавать новые модели, и дать читателю понять, что иногда решения проблем, кажущихся на первый взгляд практически неразрешимыми, все же существуют.

6.2. Сети, использующие статистический подход

До сих пор все рассмотренные нами модели нейронных сетей использовали *детерминированные* алгоритмы обучения. Слово “детерминированные” здесь означает, что при необходимости обновления весовых значений или значения активности элемента соответствующие величины изменений могут быть определены в результате непосредственного прямого вычисления. Даже в сети Хопфилда, где обновляемый элемент выбирается случайно, сами изменения являются детерминированными, поскольку если элемент уже выбран, новое значение его активности может быть вычислено вполне однозначно. В противоположность этому, для *стохастического* алгоритма обучения в любой момент времени мы остаемся в неведении относительно того, как изменится состояние сети (“состояние” в данном случае означает набор текущих весовых значений и активность элемента). Другими словами, по информации о текущем состоянии предсказать следующее состояние оказывается невозможным. Например, при использовании стохастического алгоритма обновления состояния элемента в сети Хопфилда для принятия решения о том, должен данный элемент перейти в новое вычисленное состояние или нет, используется функция распределения вероятностей. Поэтому даже когда элемент выбран для обновления, непосредственного обновления его состояния может и не произойти. Но, хотя мы и не можем сказать, как стохастическая сеть будет реагировать в каждый конкретный момент времени, мы в общем на самом деле знаем, как будет вести себя сеть по прошествии достаточно долгого периода времени.

6.2.1. Метод модельной “закалки”

Метод модельной “закалки” является методом, часто используемым при решении проблем оптимизации. Решение рассматриваемой проблемы представляется в виде решения задачи минимизации некоторой функции стоимости. Функция стоимости определяется в терминах глобальной энергии сети. Например, определенная стоимость соответствует каждому маршруту, который принимает решение пройти продавец, чтобы посетить своих клиентов в разных городах. В данном случае стоимость

может быть выражена в терминах расстояния, которое предстоит преодолеть, и тогда функция стоимости возвратит длину выбранного маршрута. Маршруты с меньшим расстоянием будут иметь меньшую стоимость. Самым оптимальным маршрутом в данном случае будет маршрут наименьшей длины. Иногда невозможно найти оптимальное решение, тогда следует искать разумно оптимальное решение.

Метод модельной “закалки” базируется на аналогии с процессом закалки металла. Когда металл подвергается закалке, он нагревается почти до точки плавления, а потом медленно охлаждается до комнатной температуры. Процесс закалки делает металл более гибким, так что становится возможным придать ему нужную форму без изломов. Когда металл нагрет до высокой температуры, атомы двигаются хаотически. Если металл быстро охладить, атомы застынут в случайных положениях. Если же металл, напротив, охлаждать медленно, атомы будут стремиться выстроиться регулярным образом. Поэтому основой всего процесса является управление графиком снижения температуры.

Аналогию с функцией оптимизации иллюстрируют рис. 6.1 и 6.2. На рис. 6.1 показана кривая, заданная некоторой гипотетической функцией одной переменной. Эта функция не является гладкой и имеет несколько (локальных) минимумов. Процедура для нахождения глобального минимума является очень простой. Сначала выбирается диапазон, в котором будет оцениваться функция. Значения функции оцениваются в некотором числе случайно выбираемых точек, при этом запоминается наименьшее из них. Диапазон затем уменьшается, и значения функции оцениваются снова в некотором числе случайно выбираемых точек. Снова сохраняется наименьшее из этих значений. Процесс повторяется снова и снова с постепенным уменьшением диапазона.

Выбор диапазона и его сокращение аналогичен установке изначально высокой температуры и постепенному уменьшению этой температуры на последующих стадиях. Сначала точка может находиться везде без ограничений, “прыгая” от одного конца кривой к другому, поскольку диапазон изначально выбирается большим. Опять же, это напоминает хаотическое движение атомов металла при высокой температуре.

Еще одним применением метода модельной “закалки” является обход локальных минимумов. Детерминированные алгоритмы типа алгоритма обратного распространения, использующие метод градиентного спуска, часто приводят к локальным минимумам. Попад в такой минимум, сеть не может больше двигаться по поверхности ошибок к более оптимальному решению. Алгоритмы, подобные градиентному спуску, всегда пытаются продвинуться вниз, чтобы уменьшить величину полной ошибки, но

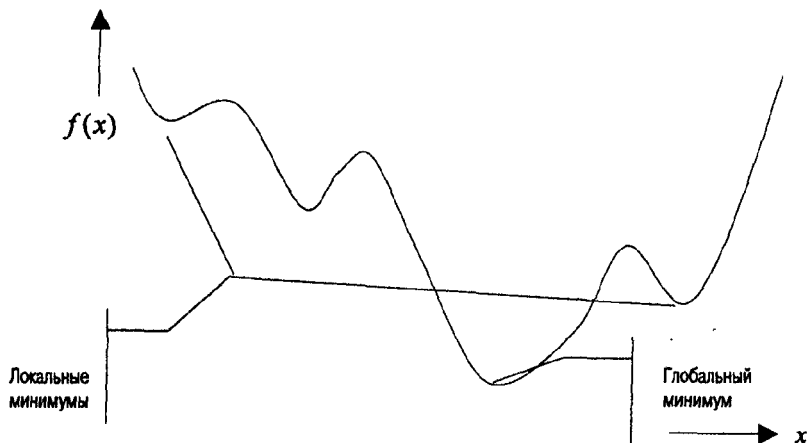


Рис. 6.1. Локальные минимумы и глобальный минимум

иногда желательно что-то и проигнорировать, чтобы открылись возможности для дальнейшего прогресса. Метод модельной “закалки” сначала ухудшает работу сети, чтобы потом ее усовершенствовать. Например, при оценке точки на кривой эта точка никогда не игнорируется, если для нее работа сети улучшается, но иногда точка может использоваться, даже если для нее сеть работает не так эффективно.

Единого и окончательного алгоритма модельной “закалки” нет, но принцип, на котором базируется метод модельной “закалки”, заложен в основу алгоритма Метрополиса (Metropolis). Основной процедурой этого алгоритма является случайный выбор части системы для изменения (например, компонента вектора). Изменения всегда принимаются, если уменьшается глобальная энергия системы, а если наблюдается рост энергии, то изменения принимаются с вероятностью p , задаваемой формулой

$$p = \exp\left(-\frac{\Delta E}{T}\right), \quad (6.1)$$

где ΔE — изменение энергии, а T обозначает температуру.

Пример алгоритма минимизации функции

Рассмотрим простой алгоритм модельной “закалки” для минимизации действительнзначной функции $f(x)$ двоичного вектора x . Предлагаемое ниже представление процедуры аналогично представлению в [Geman, Hwang, 1986].

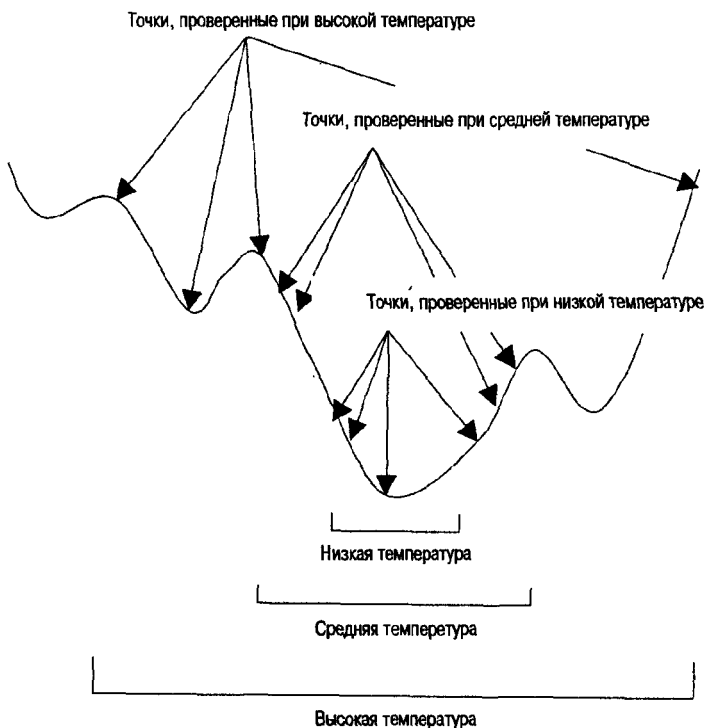


Рис. 6.2. Нахождение глобального минимума с помощью метода модельной "закалки"

Случайным образом выбирается один элемент двоичного вектора, и выбранный бит переключается (т.е. 0 заменяется на 1, а 1 — на 0). Функция оценивается с той целью, чтобы увидеть, какое значение должно быть возвращено, если принять предложенное изменение (переключение бита), и вычисляется соответствующее изменение энергии. Если энергия уменьшается, изменение принимается, иначе изменение принимается с вероятностью, заданной равенством (6.1). Формально процедура выглядит следующим образом.

1. Случайным образом выбираются начальный вектор x и начальное значение T .
2. Создается копия x с именем x_{new} , и случайным образом выбирается компонента x_{new} для изменения. Бит выбранной компоненты переключается.

3. Вычисляется изменение энергии.
4. Если изменение энергии меньше нуля, устанавливается $x = x_{\text{new}}$. Иначе с помощью функции однородного распределения плотностей вероятностей выбирается случайное число между 0 и 1. Если случайное число меньше, чем $\exp(-\Delta E/T)$, устанавливается $x = x_{\text{new}}$.
5. Если было выполнено заданное число (M) изменений вектора x , для которых значение f уменьшилось, или было выполнено N изменений x с момента последнего изменения температуры, то устанавливается $T = \alpha T$.
6. Если минимальное значение f не уменьшается уже в течение некоторого определенного фиксированного числа L последних итераций, то процесс останавливается, иначе происходит возврат к п. 2, и вычисления продолжаются.

Здесь M меньше N, а L обычно намного больше N. Понижение температуры управляется константой α , значение для которой, как правило, выбирается из диапазона от 0.8 до 0.9999. Начальное значение T выбирается так, что для всех изменений энергии имеет место неравенство $\exp(-\Delta E/T) \geq 0.9999$.

Машина Больцмана

Машина Больцмана, предложенная в [Ackley *et al.*, 1985], представляет собой нейронную сеть, использующую идею модельной “закалки” для обновления состояния сети. В своей базовой форме машина Больцмана является сетью Хопфилда (см. главу 4), использующей для обновления состояния элемента сети стохастический процесс. В предлагаемом здесь представлении состояния активности являются биполярными (т.е. характеризуются значениями +1 и -1). Энергия сети Хопфилда была определена в главе 4 следующей формулой:

$$E = -\frac{1}{2} \sum_j \sum_i s_j s_i w_{ij},$$

где s обозначает состояние элемента сети. Если элемент j изменяет состояние на величину Δs_j , то изменение энергии будет равно

$$\Delta E = -\Delta s_j \sum_i s_i w_{ij}.$$

Изменение энергии может также быть выражено в виде

$$\Delta E = -2s_j \sum_i s_i w_{ij}.$$

Это легко доказать, рассматривая возможные изменения состояния элемента. Если состояние s_j в данный момент равно -1 и оно изменяется к значению $+1$, изменение будет равно $+2$ или $-2s_j$. Если состояние s_j в данный момент равно $+1$ и оно изменяется к значению -1 , изменение будет равно -2 или $-2s_j$. Для вычисления вероятности принятия предложенного изменения состояния обычно используется функция-сигмоид:

$$p = \frac{1}{1 + \exp\left(-\frac{\Delta E}{T}\right)}.$$

Поэтому вероятность перехода элемента в новое состояние оказывается равной:

$$p = \frac{1}{1 + \exp\left(2s_j \frac{\sum_i s_i w_{ij}}{T}\right)}. \quad (6.2)$$

Машина Больцмана использовалась для решения ряда проблем оптимизации, включая проблему коммивояжера (нахождение оптимального маршрута между несколькими городами).

Машина Больцмана может иметь и скрытые элементы. Видимые элементы (т.е. элементы, используемые для сопряжения с внешней средой) могут быть разделены на входные и выходные. Пример соответствующей архитектуры показан на рис. 6.3. Все элементы имеют двунаправленные связи со всеми другими элементами, кроме элементов входного слоя, которые не связаны непосредственно с элементами выходного слоя. Сеть с такой архитектурой можно подвергнуть управляемому обучению. В ходе обучения входные и выходные элементы закрепляются (задаются входным вектором и ассоциированным с ним выходным вектором). По завершении обучения закрепляются только входные элементы, и сеть выполняет некоторое число итераций до тех пор, пока в результате не установятся значения выходных элементов.

Процесс обучения управляемой сети Больцмана разделяется на две фазы: фазу фиксации, в ходе которой входные и выходные элементы закрепляются входным и целевым образцами, и фазу свободного выполнения, в ходе которой закрепляются только входные элементы. Сетевая статистика собирается на протяжении обеих фаз и используется затем для того, чтобы обновить весовые значения.

Вес связи между элементами i и j корректируется в соответствии с формулой

$$\Delta w_{ij} = \eta(\rho_{ij}^+ - \rho_{ij}^-),$$

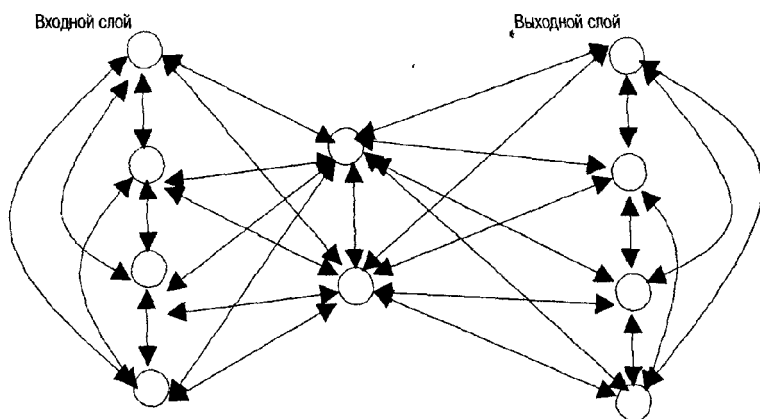


Рис. 6.3. Пример машины Больцмана с входным и выходным слоями. В данном случае сеть имеет одинаковое число входных и выходных элементов и может использоваться для автоассоциации

где ρ_{ij}^* обозначает корреляцию между элементами i и j в ходе фазы фиксации, а ρ_{ij} — корреляцию в ходе фазы свободного выполнения; в нашем случае корреляция означает вероятность одновременного нахождения двух элементов во “включенном” состоянии. Мы не будем разбираться в деталях алгоритма, но, по сути, здесь для каждой пары входных и выходных образцов закрепляются видимые элементы и выполняется модельная “закалка”. Для каждого уровня температуры происходит обновление состояний скрытых элементов в соответствии с правилом вероятностей, представленным равенством (6.2). Для конечной температуры собирается статистика, позволяющая оценить корреляции ρ_{ij}^* . Процедура повторяется для фазы свободного выполнения, по завершении которой весовые значения обновляются. Весь процесс повторяется до тех пор, пока изменения весовых значений не прекратятся (т.е. до того, как сеть сойдется).

6.2.2. Вероятностные нейронные сети

В этом разделе мы рассмотрим сеть, которую можно использовать для классификации образов и которая называется *вероятностной нейронной сетью* (сеть PNN — Probabilistic Neural Network). Сеть PNN на самом деле представляет собой параллельную реализацию давно известных статистических методов. В работе [Specht, 1990] было показано, как для клас-

сификации образцов можно использовать нейронную сеть, в которой реализуются статистические методы. Прежде чем рассмотреть детали архитектуры такой сети, выясним принципы ее работы.

В сети PNN образцы классифицируются на основе оценок их близости к соседним образцам. Расстояние до соседних образцов является важным фактором при классификации нового образца, но важными являются и особенности распределения соседних образцов. Статистические методы используют ряд критериев, на основе которых принимается решение о том, к какому из классов отнести еще неклассифицированный образец. Взгляните на рис. 6.4. Одним из методов, на основе которого можно принять решение о классе нового образца, является вычисление центроида каждого класса (т.е. усредненной характеристики размещения всех образцов в классе). Такой метод прекрасно подходит для распределений, подобных показанному на рис. 6.4а, но может приводить к ошибкам классификации при распределениях, подобных показанным на рис. 6.4б. Альтернативный простой и часто применяемый критерий классификации основан на использовании “ближайшего соседа”: для неклассифицированного образца ищется ближайший образец с известным классом, и неклассифицированный образец относится к тому же классу. Но, опять же, такой метод тоже не всегда работает, что иллюстрирует рис. 6.4в. Более “изошренные” методы в дополнение к рассмотрению расстояния учитывают плотность распределения соседних образцов.

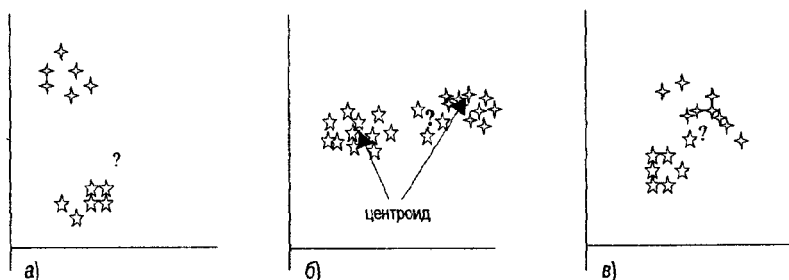


Рис. 6.4. Не использующие управление методы классификации точек. В случае а неизвестный экземпляр, обозначенный знаком “?”, кажется принадлежащим классу +; в этом случае метод центроида правильно определяет класс. В случае б неизвестный экземпляр кажется принадлежащим классу +; в этом случае метод центроида даст неверный результат. В случае в неизвестный экземпляр кажется принадлежащим классу +: ближайший “сосед” этого экземпляра не является представителем класса +. В данном случае правильная классификация зависит от учета особенностей распределения соседних образцов

В основу классификации в сети PNN положено использование методов Байеса (Bayes). Идея состоит в том, что для каждого образца можно принять решение на основе выбора наиболее вероятного класса из тех, которым мог бы принадлежать этот образец. Такое решение требует оценки функции плотности вероятностей для каждого класса. Эта оценка получается в результате рассмотрения учебных данных. Формальным правилом является то, что класс с наиболее плотным распределением в области неизвестного экземпляра будет иметь преимущество по сравнению с другими классами. Точно так же будет иметь преимущество и класс с высокой априорной вероятностью или высокой ценой ошибки классификации. Для двух классов A и B в соответствии с данным правилом выбирается класс A, если:

$$h_A c_A f_A(x) > h_B c_B f_B(x),$$

где h обозначает априорную вероятность, c — цену ошибки классификации, а $f(x)$ — функцию плотности вероятностей. Вообще говоря, оценка стоимости ошибки классификации требует хорошего знания приложения, но во многих приложениях цена ошибки классификации и априорные вероятности выбираются одинаковыми для всех классов.

Игнорирование цены ошибки и априорных вероятностей все же не избавляет от необходимости оценки функции плотности распределения вероятностей. Оценить эту функцию можно с помощью метода Парцена (Parzen), в котором используется весовая функция, имеющая центр в точке, представляющей учебный образец. Такая весовая функция называется *функцией потенциала* или *ядром*. Чаше всего в качестве ядра используется *функция Гаусса*. Форма этой функции в случае одной переменной показана на рис. 6.5. Для двух переменных (т.е. в двумерном случае) функция имеет форму колокола, как показано на рис. 6.6.

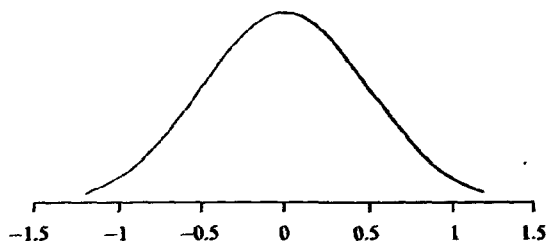


Рис. 6.5. Функция Гаусса одной переменной

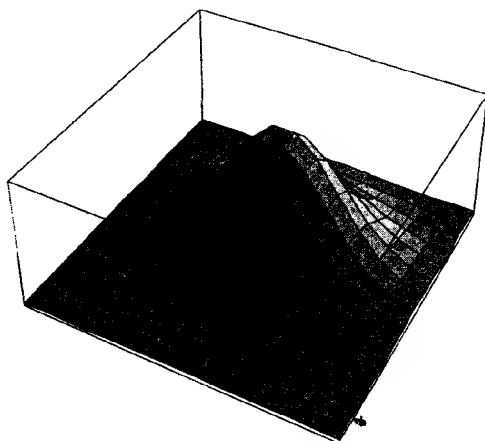


Рис. 6.6. Функция Гаусса двух переменных

Чтобы построить функцию распределения класса, для каждого учебного вектора рассматривается функция Гаусса с центром в точке, соответствующей этому вектору. Затем функции Гаусса для всех векторов суммируются, в результате чего получается нужная функция распределения. Оценка функции распределения в одномерном случае для данных, принадлежащих одному классу, показана на рис. 6.7. На этот раз использовалась более простая форма функции Гаусса, а именно:

$$g(\mathbf{x}) = \sum_{i=1}^n \exp\left(\frac{-\|\mathbf{x} - \mathbf{x}_i\|^2}{\sigma^2}\right).$$

Параметр σ задает ширину функций и определяет их влияние, что иллюстрируется на рис. 6.7–6.9.

Пример 6.1.

Имеется два класса данных одной переменной, как показано на рис. 6.10. Для образца, размещенного в точке 0.2, класс не известен. Используя функцию распределения с гауссовым ядром, определите класс, из которого взят указанный образец.

Решение 6.1.

Значение для σ принимается равным 0.1. В табл. 6.1 показаны результаты оценки плотности распределения вероятностей. Хотя неиз-

вестный экземпляр находится ближе всего к точке из класса А, вычисления заставляют отдать предпочтение классу В. Причиной, по которой В оказывается предпочтительнее, является высокая плотность точек этого класса около значения 0.35.

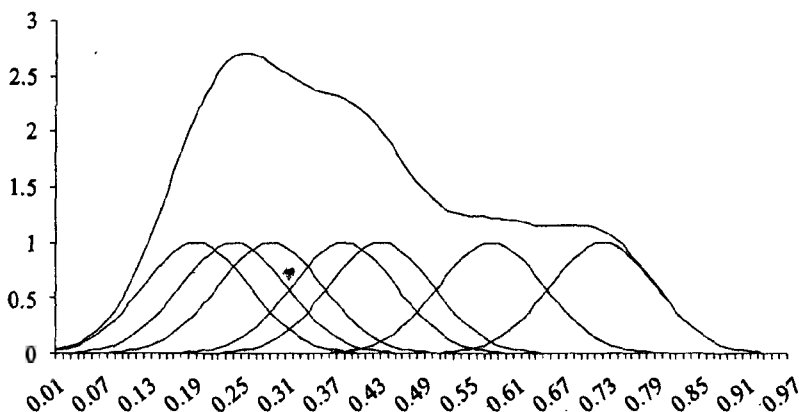


Рис. 6.7. Оценка функции распределения, полученная в виде суммы функций Гаусса с центрами в точках, соответствующих данным имеющихся образцов; здесь $\sigma = 0.1$

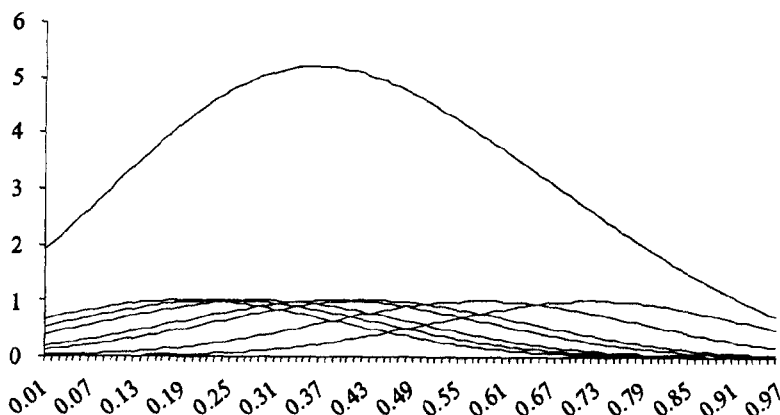


Рис. 6.8. Оценка, подобная показанной на рис. 6.7, но для $\sigma = 0.3$. Такая ширина оказывается слишком большой и есть опасность, что классы окажутся размытыми (что означает высокую вероятность ошибок классификации)

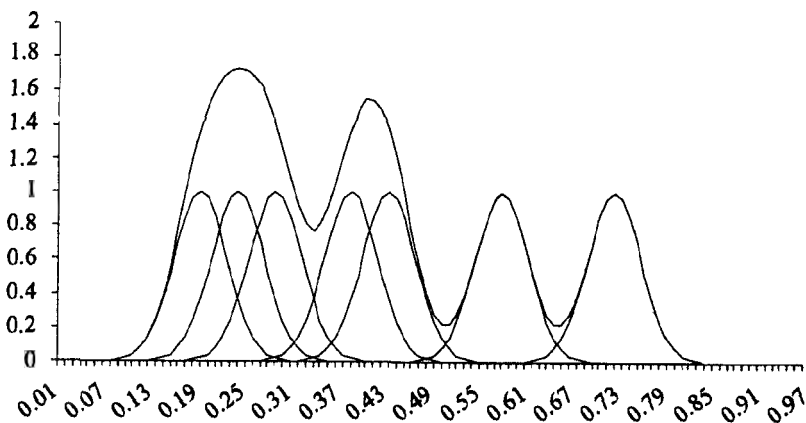


Рис. 6.9. Оценка, подобная показанной на рис. 6.7, но для $\sigma = 0.05$. Если ширина пиков становится слишком малой, есть опасность ухудшения свойств обобщения: допуски во-круг учебных образцов оказываются слишком маленькими

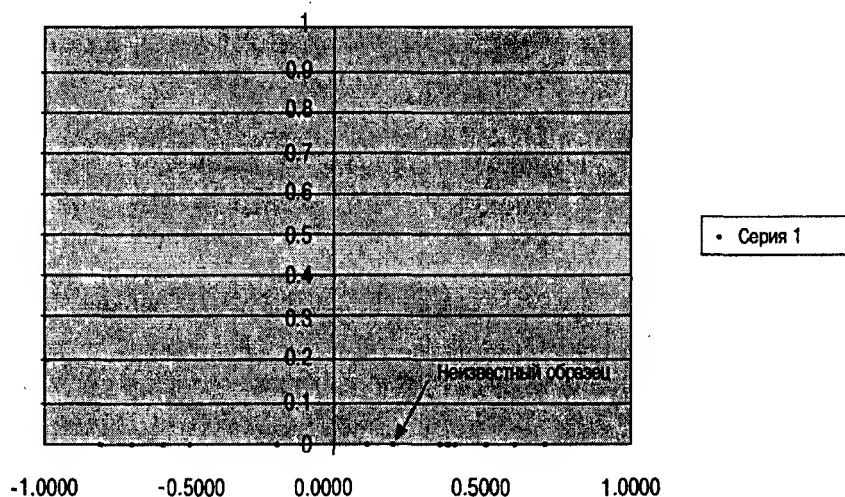


Рис. 6.10. Неизвестный экземпляр, классифицируемый с помощью функции плотности распределения вероятностей (см. пример 6.1)

Таблица 6.1. Результаты вычисления оценок плотности распределения вероятностей для примера 6.1.

Класс	Учебный образец	Расстояние до неизвестного	Плотность распределения
A	-0.2000	0.1600	0.0000
A	-0.5000	0.4900	0.0000
A	-0.6000	0.6400	0.0000
A	-0.7000	0.8100	0.0000
A	-0.8000	1.0000	0.0000
A	0.1000	0.0100	0.3679
			0.3679
B	0.3500	0.0225	0.1054
B	0.3600	0.0256	0.0773
B	0.3800	0.0324	0.0392
B	0.3650	0.0272	0.0657
B	0.3550	0.0240	0.0905
B	0.4000	0.0400	0.0183
B	0.5000	0.0900	0.0001
B	0.6000	0.1600	0.0000
B	0.7000	0.2500	0.0000
			0.3965

Архитектура нейронной сети PNN

Пример архитектуры сети для решения простой задачи показан на рис. 6.11. Задачей входного слоя является распределение данных входного образца для слоя образцов. В данном случае каждый входной набор данных имеет четыре признака. Слой образцов имеет по одному элементу для каждого образца из набора учебных данных. Входной слой и слой образцов образуют полносвязную структуру. Для входящих в элемент слоя образцов связей весовые значения устанавливаются равными элементам соответствующего вектора-образца. Например, для первого элемента слоя образцов значение его первого входящего веса будет установ-

лено равным значению первого элемента первого вектора-образца, значение второго входящего веса — второму элементу вектора и т.д. Активность элемента слоя образцов будет равна

$$O_j = \exp\left(\frac{-\sum (w_{ij} - x_i)^2}{\sigma^2}\right), \quad (6.3)$$

где x обозначает неизвестный входной образец. В этом выражении используется квадрат евклидова расстояния от неизвестного экземпляра до элемента слоя образцов.

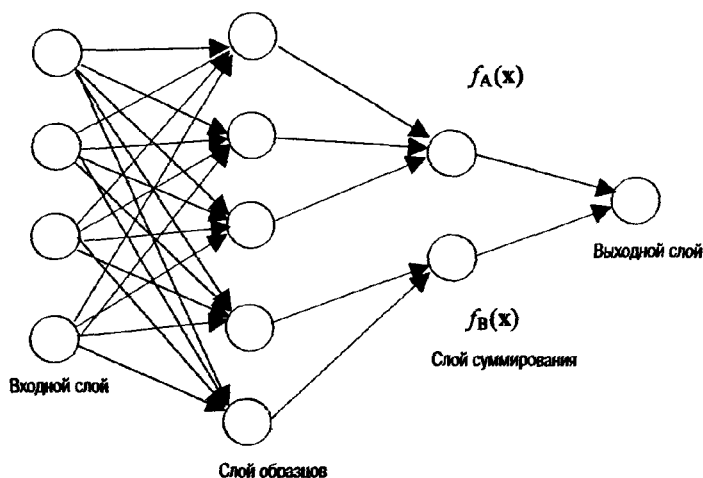


Рис. 6.11. Пример архитектуры сети PNN

Слой суммирования имеет по одному элементу для каждого класса из учебного множества данных. К любому элементу слоя суммирования идут связи только от элементов слоя образцов, принадлежащих соответствующему классу. Весовые значения связей, идущих от элементов слоя образцов к элементам слоя суммирования, фиксируются равными 1. Элемент слоя суммирования просто суммирует выходные значения элементов слоя образцов. Эта сумма дает оценку значения функции плотности распределения вероятностей для совокупности экземпляров соответствующего класса. Выходной элемент представляет собой дискриминатор пороговой величины, указывающий элемент слоя суммирования с максимальным значением активности (т.е. указывает класс, к которому принадлежит неизвестный экземпляр).

Для сети PNN не требуется обучения в том смысле, какое требуется для сетей с обратным распространением ошибок, так как все параметры сети PNN (число элементов и значения весов) определяются непосредственно учебными данными.

Процедура для использования сети PNN является относительно простой. Архитектура сети определяется структурой учебных данных:

- число входных элементов равно числу признаков;
- число элементов слоя образцов равно числу учебных образцов;
- число элементов слоя суммирования равно числу классов.

Первый слой весовых значений определяется учебными образцами. Для второго слоя все значения устанавливаются равными единице. Весовые значения конечного слоя устанавливаются так, чтобы на выходе распознавался элемент слоя суммирования с наибольшим значением активности. Для элементов слоя образцов необходимо выбрать подходящую функцию активности. Как правило, используется ядро Гаусса в виде, показанном в выражении (6.3). Значение σ задает ширину функции активности. Значение σ оказывается очень важным, поэтому чаще всего оно подбирается в результате эксперимента.

После того как сеть построена, неизвестный экземпляр можно подать на вход сети, и в результате прямого прохода через сеть выходной слой укажет класс, к которому, вероятнее всего, принадлежит образец.

Данное здесь представление сети PNN требует вычисления евклидова расстояния от неизвестного экземпляра до всех учебных образцов. Если все входные векторы имеют единичную длину, то функция активности для элемента слоя образцов может быть представлена в форме, содержащей более удобную для использования сумму произведений:

$$O_j = \exp\left(\frac{\sum x_i w_{ij} - 1}{\sigma^2}\right).$$

Пример 6.2.

На рис. 6.12 показаны состоящее из трех классов множество учебных точек и неизвестный экземпляр. Нормализуйте данные ввода к единичной длине и, используя сеть PNN, найдите класс, к которому следует отнести неизвестный экземпляр.

Решение 6.2.

Нормализованные векторы показаны на рис. 6.13. В табл. 6.2 представлены учебные данные до и после нормализации. В табл. 6.3 представлен

неизвестный образец. В табл. 6.4 показаны результаты прохода через сеть с целью классификации вектора неизвестного образца. Значение σ равно 0.1. Вычисления помещают неизвестный образец в класс А.

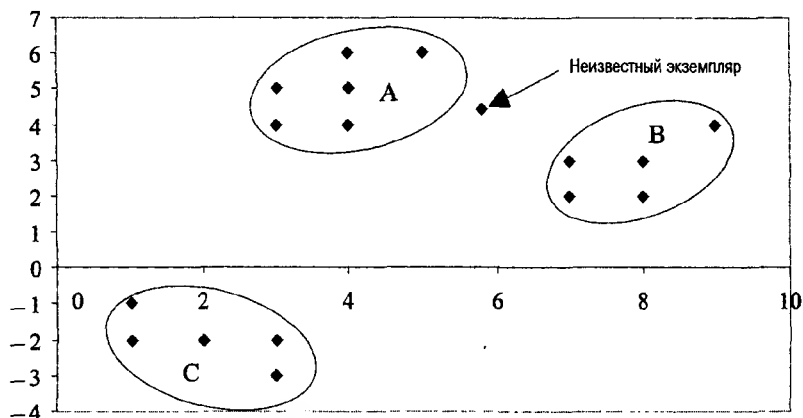


Рис. 6.12. Неизвестный экземпляр для классификации с помощью сети PNN (см. пример 6.2)

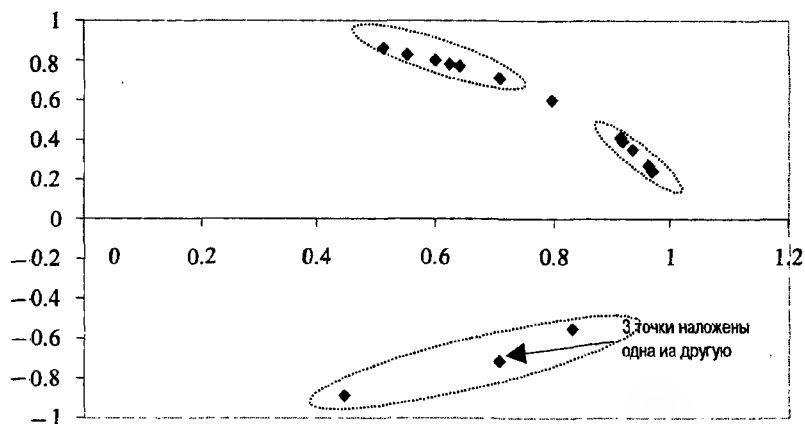


Рис. 6.13. Результат нормализации векторов, показанных на рис. 6.12

Таблица 6.2. Нормализованные учебные данные

Ненормализованный вектор		Нормализованный вектор	
x_1	x_2	x_1	x_2
3.0000	5.0000	0.5145	0.8575
4.0000	4.0000	0.7071	0.7071
3.0000	4.0000	0.6000	0.8000
5.0000	6.0000	0.6402	0.7682
4.0000	6.0000	0.5547	0.8321
4.0000	5.0000	0.6247	0.7809
7.0000	2.0000	0.9615	0.2747
7.0000	3.0000	0.9191	0.3939
8.0000	2.0000	0.9701	0.2425
8.0000	3.0000	0.9363	0.3511
9.0000	4.0000	0.9138	0.4061
1.0000	-1.0000	0.7071	-0.7071
1.0000	-2.0000	0.4472	-0.8944
2.0000	-2.0000	0.7071	-0.7071
3.0000	-2.0000	0.8321	-0.5547
3.0000	-3.0000	0.7071	-0.7071

Таблица 6.3. Неизвестный экземпляр

Ненормализованный		Нормализованный	
x_1	x_2	x_1	x_2
5.8000	4.4000	0.7967	0.6044

Таблица 6.4. Вычисления сети PNN для классификации неизвестного экземпляра, представленного в табл. 6.3

Элемент слоя образцов	w_1	w_2	Активность	Активность элемента слоя суммирования
1	0.5145	0.8575	0.0008	
2	0.7071	0.7071	0.3950	

Окончание табл. 6.4

Элемент образцов	слоя	w_1	w_2	Активность	Активность элемента слоя суммирования
3		0.6000	0.8000	0.0213	
4		0.6402	0.7682	0.0768	
5		0.5547	0.8321	0.0040	
6		0.6247	0.7809	0.0480	0.5459
7		0.9615	0.2747	0.0011	
8		0.9191	0.3939	0.0516	
9		0.9701	0.2425	0.0003	
10		0.9363	0.3511	0.0153	
11		0.9138	0.4061	0.0706	0.1389
12		0.7071	-0.7071	0.0000	
13		0.4472	-0.8944	0.0000	
14		0.7071	-0.7071	0.0000	
15		0.8321	-0.5547	0.0000	
16		0.7071	-0.7071	0.0000	0.0000

Сети PNN могут быть более сложными, чем представленная здесь сеть. Например, для каждого входного признака можно использовать разные значения σ . Таким образом оказывается возможным в значительной степени контролировать форму классификационной поверхности, соответствующей учебным образцам.

Сети PNN очень удобно использовать для классификации. Они быстро обучаются, допускают наличие ошибочных данных и обеспечивают полезные результаты даже на малых наборах учебных данных. Но сети PNN оказываются весьма требовательными в отношении ресурсов. Решение некоторых проблем требует сотен и даже тысяч учебных образцов, в результате чего классификация каждого неизвестного экземпляра потребует немало времени. Однако необходимо помнить, что если сеть реализована в виде аппаратных средств, то вычисления чаще всего выполняются параллельно.

Сеть PNN не является столь общей, как некоторые другие нейронные сети. Так, в своей базовой форме сеть PNN ограничивается задачами классификации, в отличие от многослойной сети с прямой связью и обратным распространением ошибок, которая может моделировать отображение общего вида. Тем не менее классификация требуется во многих задачах, а классификацию сети PNN выполняют очень хорошо. Не случайно в [Masters, 1995] автор заявляет, что сеть PNN является его любимой нейронной сетью.

6.3. Пример модульной нейронной сети

Сеть BP-SOM представляет собой сеть, объединяющую многослойную сеть с прямой связью (сеть MFN — Multilayered Feedforward Network), обученную по алгоритму обратного распространения ошибок (backpropagation), и самоорганизующуюся карту признаков (сеть SOM — Self-Organizing Map). Мы рассматриваем сеть BP-SOM для того, чтобы понять, как, не затрачивая особых усилий, строить сети с новой архитектурой, чтобы подчеркнуть, что не следует спешить отказываться от использования нейронных сетей, если при первых попытках сеть не смогла справиться с поставленной перед ней задачей.

По своей архитектуре сеть BP-SOM является стандартной сетью с прямой связью, имеющей один или несколько скрытых слоев (для более полной информации по этому вопросу см. [Weijters *et al.*, 1997]). С каждым скрытым слоем ассоциируется сеть SOM. В следующем описании мы предполагаем наличие только одного скрытого слоя, но точно такая же процедура используется и при наличии нескольких скрытых слоев.

Число элементов в сети SOM выбирается произвольным образом, а число входов каждого элемента сети SOM устанавливается равным числу элементов в соответствующем скрытом слое. Сети MFN и SOM обучаются параллельно. Значения активности скрытого слоя служат входными данными для сети SOM. В ходе обучения сеть SOM самоорганизуется, и эта самоорганизация переводится в классификацию после назначения каждому элементу сети SOM метки класса. Метка класса элемента сети SOM определяется следующим образом. После некоторого числа учебных циклов все учебные образцы (вместе с их известными выходными классами) один за другим предъявляются сети MFN. Значения активности скрытого слоя подаются на вход сети SOM и элементу-победителю сети SOM (т.е. элементу, оказывающемуся в метрике Евклида ближе всех других к вектору значений активности скрытого слоя). С каждым элементом сети SOM связываются счетчики выходных классов. Каждый раз, когда элемент SOM оказывается победителем в конку-

ренции за учебный образец, счетчик соответствующего этому образцу класса увеличивается на единицу. После рассмотрения всех учебных образцов каждый элемент сети SOM получает метку, соответствующую классу с наивысшим значением счетчика. Вместе с меткой класса рассматривается и значение надежности, представляющее собой отношение значения счетчика соответствующего класса к общему числу побед данного элемента. Пример архитектуры такой сети показан на рис. 6.14.

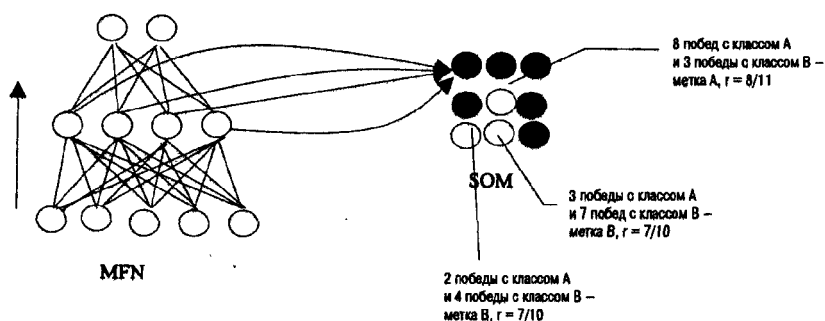


Рис. 6.14. В данном примере имеется 27 учебных образцов – 13 из класса А и 14 из класса В. Чтобы избежать путаницы, здесь показаны связи сети MFN только с одним элементом сети SOM

При корректировке весов в ходе обучения сети MFN рассматриваются два вектора ошибок. Первый из этих векторов вычисляется с использованием стандартной техники обратного распространения ошибок. Это значит, что образец подается на рассмотрение сети MFN, и реальный выход сети сравнивается с плановым выходом. Выходная ошибка затем используется для того, чтобы вычислить значения ошибок для каждого из скрытых элементов (v_{bp_error}). Второе значение ошибки вычисляется для сети SOM. Значения активности скрытых элементов, генерируемые текущим входным образцом, сравниваются со значениями элемента-победителя сети SOM, имеющего метку класса, соответствующего текущему образцу. Другими словами, элементы SOM конкурируют за вектор значений активности скрытого слоя, но в конкуренции участвуют только элементы с правильной меткой класса. Разность между вектором значений активности скрытого слоя и вектором весовых значений элемента-победителя сети SOM и является ошибкой сети SOM (v_{som_error}). Если элемента сети SOM с меткой класса текущего образца не найдено, все элементы вектора v_{som_error} устанавливаются равными нулю. Ошибка скрытого элемента вычисляется по формуле

$$v_{bp_som_error_j} = \begin{cases} ((1-\alpha) \times v_{bp_error_j}) + (r \times \alpha \times v_{som_error_j}), & \text{если } r > t, \\ v_{bp_error_j}, & \text{в противном случае,} \end{cases}$$

где α обозначает влияние ошибки сети SOM, которое обычно устанавливается равным 0.25 (если установить его равным нулю, то будет иметь место стандартное обратное распространение ошибок), r является показателем надежности, а t — пороговым значением, обычно равным 0.95.

Процесс обучения не отличается от процесса обучения стандартной сети с обратным распространением ошибок, если не учитывать добавление дополнительного учитывающего ошибку члена для каждого скрытого элемента. Процесс обучения формально можно представить в следующем виде.

1. Весовые значения для сетей MFN и SOM устанавливаются равными случайным значениям. Для всех элементов SOM устанавливаются значения меток, соответствующие неизвестному классу, а значения счетчиков классов и надежности устанавливаются равными нулю.
2. Сеть MFN обучается в течение некоторого фиксированного числа эпох (m). На протяжении каждой эпохи выполняется следующее.
 - Для всех учебных образцов и соответствующих им выходных классов вычисляются векторы значений активности скрытого слоя, и эти векторы используются для обучения сети SOM; вычисляется $v_{bp_som_error}$ и обновляются весовые значения сети MFN.
 - После каждых n циклов ($1 < n \leq m$) метки классов и показатели надежности элементов сети SOM вычисляются снова.

По завершении обучения сеть MFN может работать в автономном режиме.

Преимуществом использования сети BP-SOM являются ее вышеупомянутые исключительные способности к обобщению. Мы знаем, что хорошие способности обобщения важны для любого приложения, поскольку от сети будет мало пользы, если она не сможет работать с данными, которые не предъявлялись ей во время обучения. Бывает так, что сеть MFN, обучавшаяся с относительной легкостью по методу обратного распространения ошибок, оказывается неспособной к обобщению при приемлемом уровне производительности. Иногда улучшить свойства обобщения помогают очень простые меры (например, проверка на перетренированность), но очень часто решение не будет столь простым. Когда сеть работает неадекватно, возникает искушение квалифицировать сеть, как не способную выполнить поставленную перед ней задачу. В работе [Clark, 1993] описан ряд экспериментов, проведенных разными исследователями, чьи сети сначала не смогли выполнить свои задачи, но улучшили качество своей работы после изменения структуры учебного набо-

ра. Структура обычно меняется в результате декомпозиции задачи. Это можно сделать с помощью разбиения учебных данных на фазы (обучение на подмножестве учебных данных перед рассмотрением остальных учебных экземпляров) или с помощью разбиения задачи на подзадачи, которые можно пытаться решить, используя разные сети.

Одним из экспериментов, о которых сообщает Кларк (Clark), является попытка Норриса (см. [Norris, 1989]) обучить сеть MFN выводу дня недели по данной дате. Например, по дате “24 февраля 1998 г.” сеть должна выдать ответ — “вторник”.

Сначала в эксперименте Норриса рассматривалась сеть MFN, обучаемая по алгоритму обратного распространения ошибок. Сеть не могла обобщать полученные данные даже при варьировании числа слоев и элементов. Норрис провел логический анализ алгоритма вычисления дня недели по данной дате. Процедура вычисления даты была разбита на три шага. Например, можно выбрать базовый месяц (скажем, ноябрь 1957 г.) и указать день для каждого числа базового месяца. Затем обучить сеть находить решение для чисел всех месяцев базового года (1957), используя смещение относительно базового месяца. Наконец, обучить сеть находить решение по смещению между годами (например, принимается смещение на один день для следующего года). Каждый шаг моделировался соответствующей подсетью. Первая подсеть завершила обучение до начала обучения второй подсети, а вторая подсеть завершила обучение до начала обучения третьей. Обобщение имело точность около 90%, а наибольшее число ошибок выпадало на високосный год.

Хотя Норрис в конечном счете создал успешно работающую сеть, это произошло после “вмешательства человека”. Такое решение зависит от хорошего знания специфики задачи. Пока что получаемые на практике решения реальных проблем используют знания экспертов, но важно выяснить, как можно было бы усовершенствовать работу самой нейронной сети. Чаше всего нейронные сети применяются вслепую: мы не знаем, как решить некоторую проблему, и поэтому пробуем использовать нейронную сеть. При этих условиях, если сети не удастся решить проблему, то мы не будем знать причин, по которым она не работает, — из-за принципиальной невозможности решить поставленную задачу, неподходящей структуры учебного набора данных или по какой-либо другой причине. Специалист по применению нейронных сетей знает, что для решения реальных проблем обычно требуется множество экспериментальных данных, но все это время делаются попытки достичь существенного прогресса с помощью разработки новых моделей нейронных сетей и понимания того, как именно нейронные сети решают конкретные проблемы.

В [Weijters *et al.*, 1997] задача выяснения дня недели по дате решена с помощью сети BP-SOM. Качество работы представленной сети является впечатляющим. Для сравнения авторы обучили и стандартную сеть MFN, используя метод обратного распространения ошибок. В ходе тестирования сеть с обратным распространением ошибок работала неадекватно в 61% случаев, в то время как для сети BP-SOM этот показатель составил всего 3%.

В главе 8 мы рассмотрим еще несколько примеров модульной архитектуры.

6.4. Практические вопросы обучения нейронных сетей

6.4.1. Выбор модели сети

Первым вопросом, на который приходится отвечать при попытках практического применения нейронных сетей, является вопрос о типе выбираемой модели. Ответ на него зависит от того, известна или нет классификация учебных данных. Например, обучение системы, распознающей клиентов с низким показателем кредитоспособности, скорее всего будет включать управляемое обучение, ввиду того, что кредитор наверняка будет иметь записи о том, как аккуратно выплачивались долги данным клиентом в прошлом. Иногда информации относительно класса, к которому следует отнести данные, нет вообще, а иногда классификация является нечеткой. Например, часто бывает трудно дать точную характеристику состояния машины, например вертолета. В настоящее время проблеме мониторинга технического состояния вертолетов уделяется много внимания. Бортовые датчики регистрируют информацию, которая затем загружается в базу данных на земле для анализа. Предполагая, что вертолет обычно находится в хорошем техническом состоянии, загруженная информация может анализироваться на предмет обнаружения существенных отклонений от параметров предыдущих полетов. Если в данных обнаруживаются большие расхождения, необходимо внимательно проверить летательный аппарат на возможность отказа. В таких ситуациях часто используется кластерный анализ. Самоорганизующаяся карта признаков Кохона при этом является не предполагающей управление нейронной сетью, имеющей много общего со статистической кластеризацией.

Природа проблемы, как правило, ограничивает широту выбора сети одной-двумя моделями. Иногда конечный выбор зависит от личных предпочтений или знаний специалиста, делающего выбор. В других случаях сама проблема снова ограничивает выбор. Предположим, например,

что система, в которой сеть реализована в программном виде, обучается по методу управляемого обучения, и задача требует, чтобы по окончании обучения время выполнения было минимальным. Для управляемого обучения подходят и сеть с обратным распространением ошибок, и вероятностная нейронная сеть. Но если проблема предполагает анализ очень большого набора данных с большим числом признаков, то вероятностная сеть может предъявить слишком большие требования к операционной системе, в которой реализуется соответствующая программа. Рассмотрим, например, учебное множество из 2000 образцов, каждый из которых имеет 30 признаков. Первый слой весов в сети с обратным распространением ошибок, имеющей 15 скрытых элементов, потребует выполнения 30×15 операций умножения для обработки неизвестного экземпляра, тогда как вероятностная сеть — 30×2000 .

6.4.2. Архитектура

Многие аспекты сетевой архитектуры (например, число элементов) часто определяются самой проблемой. Например, вероятностная нейронная сеть должна иметь архитектуру, которая практически однозначно определяется учебными данными. Число входных и выходных элементов сети с прямой связью и обратным распространением ошибок тоже диктуется рассматриваемой проблемой (числом входных признаков и числом известных классов). Размеры скрытого слоя обычно находятся экспериментально. Обычно начинают с одного скрытого слоя, который содержит 30–50% числа элементов входного слоя. Для самоорганизующейся карты Кохонена знание проблемы может подсказать общее число элементов в сети. Например, связь между набором данных и их классификацией может быть неизвестной, но обычно имеется некоторая информация относительно того, сколько должно быть классов (например, это могут быть типы отказов, которые возможны для таких машин). Обычно на каждый класс приходится по несколько элементов, что позволяет классам разместиться в разных областях пространства входных образцов.

6.4.3. Данные

Доступность и целостность данных составляют наиболее важный фактор успешного обучения нейронной сети. Данные должны в полной мере представить все возможные состояния решаемой проблемы, их должно быть достаточно для того, чтобы из них можно было извлечь данные для тестирования и проверки правильности их обработки сетью.

Данные, выбираемые для обучения, должны характеризовать все пространство, которое может занять соответствующий класс. Например, если

два класса размещены очень близко один от другого, как показано на рис. 6.15, то важно рассмотреть данные, близкие к границе, разделяющей классы, чтобы быть уверенными в том, что сеть “посвятит” часть своих ресурсов данным из этой области, поскольку в противном случае экземпляры из этой области данных могут классифицироваться с ошибками.

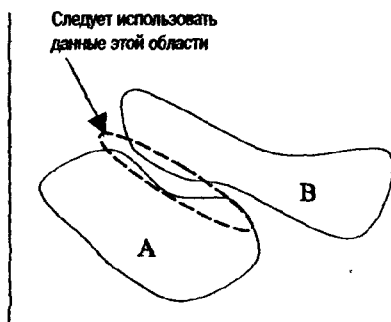


Рис. 6.15. Чтобы правильно классифицировать неизвестные данные, сеть должна обучаться на характерных данных. Чтобы распознать границу между классами А и В, сеть должна быть обучена на данных из области, указанной пунктиром

Данные следует проверить и на согласованность. Как правило, в большом наборе данных всегда находятся несогласованности и ошибки. Иногда неверные данные вводятся оператором, а некоторые значения для отдельных образцов могут быть пропущены. Иногда в данных содержатся посторонние значения. Посторонними являются точки, которые сильно выделяются из остальных данных и могут указывать на какие-то ошибки, появившиеся из-за неправильной записи исходных данных. Здесь требуется внимательность, так как иногда посторонние значения могут порождаться и полезной информацией (например, информацией об отказе машины). Даже когда используется управляемое обучение, может оказаться разумным сначала провести кластеризацию данных, используя методы обучения без управления или какой-то другой тип статистического анализа, чтобы выявить потенциальные проблемы с данными. Тогда можно будет скорректировать проблемные данные или просто отфильтровать их из учебного набора.

Тестовые данные и данные для проверки правильности должны выбираться случайным образом и, опять же, они должны быть характерными

для исследуемой проблемы. Проверка правильности данных часто оказывается полезной потому, что тестовые данные иногда используются для контроля обучения сети (см. раздел 6.4.5).

Данные обычно требуют масштабирования, чтобы они попадали в область действия сети. Например, целевые выходные данные для сети с обратным распространением ошибок и с сигмоидальной функцией активности элементов должны лежать между 0 и 1, поскольку соответствующей является область значений сигмоидальной функции. Каждый признак может требовать своего масштабирования. Предположим, что один признак изменяется в диапазоне от 300 до 2000, а другой — от 5 до 130 (скажем, это рост в миллиметрах и вес в килограммах). Тогда первый признак будет стремиться доминировать над вторым, поскольку будет иметь большее влияние на ввод соответствующего элемента. Простейшим методом масштабирования является деление значения признака на максимальное значение этого признака. В результате наибольшее значение будет ограничено значением 1. Иногда необходимо масштабировать данные. Например, для функции активности типа “сигмоид” часто используют диапазон между 0.1 и 0.9, что помогает избежать перехода сети в состояние “холостого хода” при достижении крайних пределов области ее работоспособности. Признак может быть масштабирован к диапазону 0.1–0.9 с помощью формулы

$$y = \frac{0.9 - 0.1}{x_{\max} - x_{\min}} x + \left(0.9 - \frac{0.9 - 0.1}{x_{\max} - x_{\min}} x_{\max} \right),$$

где y обозначает новое значение, а x — первоначальное значение. Например, предположим, что значения признака изменяются от 2 до 20. Значение 2 перейдет в значение

$$y = \frac{0.9 - 0.1}{20 - 2} 2 + \left(0.9 - \frac{0.8}{20 - 2} 20 \right) = 0.1,$$

которое будет новым минимальным значением. Значение 20 отобразится в 0.9, а значению 5 будет соответствовать

$$y = \frac{0.9 - 0.1}{20 - 2} 5 + \left(0.9 - \frac{0.8}{20 - 2} 20 \right) = 0.23.$$

Вместо значений 0.9 и 0.1 можно выбрать другие, если требуется масштабировать данные к другому диапазону.

Разные признаки могут также иметь существенно различные вариации в распределении значений. Например, два признака могут иметь одинаковые области изменения значений, но значения одного из признаков могут

в большинстве своем находиться около максимального значения в конце диапазона, тогда как значения другого признака могут более или менее равномерно распределяться по всему диапазону. Соответствующая иллюстрация для 10 точек показана на рис. 6.16. Другой формой масштабирования является вычисление для каждого признака среднеквадратичного отклонения и среднего значения. Имеющиеся значения признака затем масштабируются с помощью вычитания среднего и деления результата на значение среднеквадратичного отклонения для данного признака.

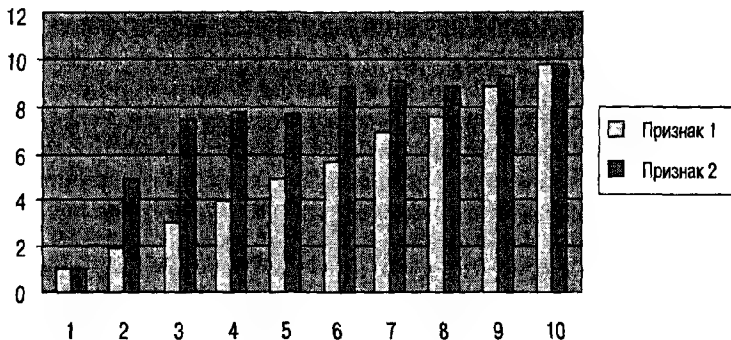


Рис. 6.16. Два признака с одинаковыми областями значений, но разными распределениями

Для проблем с множеством свойств часто наблюдается избыточность данных. Избыточность означает, что можно описать данные, используя меньше признаков, но сохранив при этом всю информацию. Эта идея может показаться странной, но ее можно иллюстрировать с помощью простого примера. Соответствующие значениям x и y из табл. 6.5 точки показаны на рис. 6.17. Эти точки лежат на прямой. Если мысленно повернуть оси так, что ось абсцисс расположится вдоль прямой, на которой лежат точки данных, то ось абсцисс можно считать новой осью, относительно которой все точки данных имеют значение 0 для их компоненты y . Другими словами, теперь для описания данных требуется только одна новая ось x . Соответствующая трансформация может быть выполнена с помощью умножения значений x и y на $1/\sqrt{2}$ и последующего их суммирования. Вместо описания данных с помощью двух признаков мы теперь имеем только один. Такая трансформация сохраняет пространственную взаимосвязь данных. Например, евклидово расстояние между точкой (3, 3) и точкой (7, 7) равно расстоянию между преобразованными точками 4.2426 и 9.8995 соответственно.

Таблица 6.5. Данные, описанные двумя переменными x и y , преобразуются так, чтобы та же информация обеспечивалась одной переменной

x	y	Преобразованное значение
1.0000	1.0000	1.4142
2.0000	2.0000	2.8284
3.0000	3.0000	4.2426
4.0000	4.0000	5.6569
5.0000	5.0000	7.0711
6.0000	6.0000	8.4853
7.0000	7.0000	9.8995
8.0000	8.0000	11.3137
9.0000	9.0000	12.7279

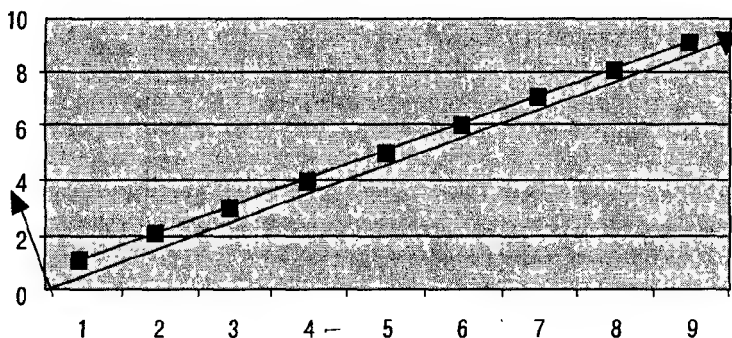


Рис. 6.17. Данные из табл. 6.15 соответствуют точкам (x, y) , расположенным на прямой. Если данные представить в новых координатах, показанных здесь осями со стрелками, для их описания будет достаточно одной координаты

Избыточность может быть выявлена с помощью анализа главных компонент. Об анализе главных компонент уже шла речь в главе 4, но ввиду важности этого метода мы обсудим его здесь снова. С помощью этого метода находится альтернативное множество осей, относительно которых может быть представлен имеющийся набор данных. Кроме того, выясняется, вдоль какой оси наблюдается наибольшая вариация данных. Например, на вышеупомянутой иллюстрации вся вариация происходит вдоль новой оси абсцисс, а отклонение вдоль новой оси

ординат оказывается равным нулю (именно поэтому вторая ось и не требуется). После того как избыточная информация удаляется, можно удалить и некоторые малозначимые компоненты (принято называть новые оси компонентами), вдоль которых наблюдается очень малая вариация данных. В результате исключения из рассмотрения малозначимых компонент некоторая информация будет утеряна, но анализ главных компонент покажет, как много информации все же будет сохранено. Вы можете, например, отбросить малозначимые компоненты, но сохранить достаточно компонент, чтобы описать 95% данных. На рис. 6.18 показанные там две главные компоненты имеют разные вариации (или длины). Первая компонента “растянута” в направлении наибольшего разброса данных, а вторая направлена под углом в 90 градусов к первой. Вторая компонента описывает остаточную вариацию, т.е. вариацию, оставшуюся после учета первой компоненты.

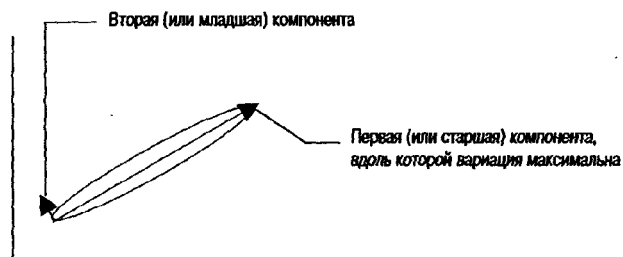


Рис. 6.18. Данные, размещенные внутри овальной области, могут быть представлены новым множеством главных осей

Автоассоциативная сеть с обратным распространением ошибок выполняет своего рода анализ главных компонент. При решении некоторых проблем может оказаться полезным выполнение предварительной обработки данных с помощью автоассоциативной сети, прежде чем использовать другую сеть. Вторая сеть должна получать на вход значения активности скрытых элементов первой сети. Но, удаляя малозначимые компоненты, нужно проявлять осторожность, так как для некоторых проблем такие малозначимые компоненты могут содержать информацию, являющуюся очень важной с точки зрения решения задачи.

Внимание должно уделяться и тому, чтобы в учебный набор данных не было внесено смещение из-за неадекватного представления признаков. В сети BP-SOM, описанной в разделе 6.3, авторы работы [Weijters *et al.*, 1997] использовали 12 входных элементов, чтобы представить месяцы

года. Разные элементы представляли разные месяцы. При наивном подходе можно было бы использовать один элемент и нумеровать месяцы от 1 до 12. Такая нумерация порождает смещение, в результате которого один месяц может иметь большее влияние, чем другой. Но в каком смысле декабрь может быть более важным, чем январь, с точки зрения вычисления данных?

6.4.4. Локальные минимумы

Для сети с обратным распространением ошибок не является редкостью попадание в локальный минимум. Локальный минимум может быть близким, но все же недостаточно близким к удовлетворительному решению проблемы. Метод градиентного спуска работает прекрасно, пока наклон функции ошибок все время ведет вниз, но на практике поверхность ошибок для большой сети будет иметь множество ложбин, холмов и складок. При таком рельефе будет достаточно мест, где можно застрять.

Проблему локальных минимумов может помочь преодолеть инерционный член, поскольку при его использовании изменения весовых значений включают часть предыдущего изменения. Если предыдущее изменение весового значения означало большой по величине шаг вниз по поверхности ошибок и сеть оказалась в локальном минимуме, то такое предыдущее изменение весового значения может помочь сети продолжить движение вниз, поскольку инерции окажется достаточно для того, чтобы преодолеть небольшую ложбину. Идея использования инерции иллюстрируется на рис. 6.19.

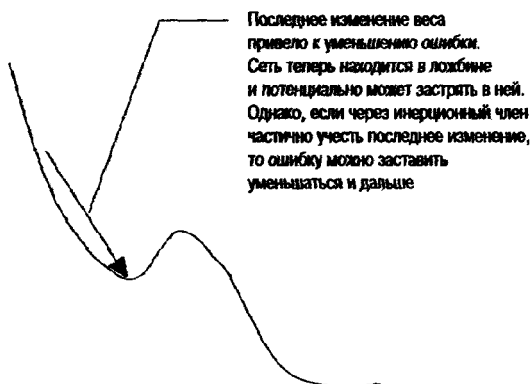


Рис. 6.19. Попытка обойти локальный минимум с помощью инерции

ординат оказывается равным нулю (именно поэтому вторая ось и не требуется). После того как избыточная информация удаляется, можно удалить и некоторые малозначащие компоненты (принято называть новые оси компонентами), вдоль которых наблюдается очень малая вариация данных. В результате исключения из рассмотрения малозначащих компонент некоторая информация будет утеряна, но анализ главных компонент покажет, как много информации все же будет сохранено. Вы можете, например, отбросить малозначащие компоненты, но сохранить достаточно компонент, чтобы описать 95% данных. На рис. 6.18 показанные там две главные компоненты имеют разные вариации (или длины). Первая компонента “растянута” в направлении наибольшего разброса данных, а вторая направлена под углом в 90 градусов к первой. Вторая компонента описывает остаточную вариацию, т.е. вариацию, оставшуюся после учета первой компоненты.

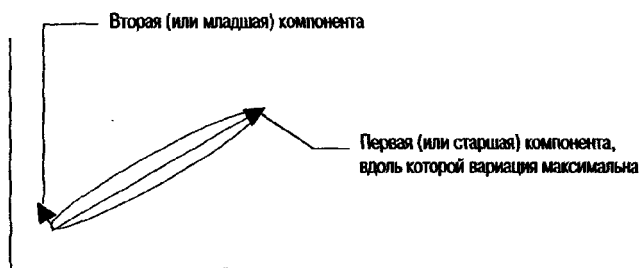


Рис. 6.18. Данные, размещенные внутри овальной области, могут быть представлены новым множеством главных осей

Автоассоциативная сеть с обратным распространением ошибок выполняет своего рода анализ главных компонент. При решении некоторых проблем может оказаться полезным выполнение предварительной обработки данных с помощью автоассоциативной сети, прежде чем использовать другую сеть. Вторая сеть должна получать на вход значения активности скрытых элементов первой сети. Но, удаляя малозначащие компоненты, нужно проявлять осторожность, так как для некоторых проблем такие малозначащие компоненты могут содержать информацию, являющуюся очень важной с точки зрения решения задачи.

Внимание должно уделяться и тому, чтобы в учебный набор данных не было внесено смещение из-за неадекватного представления признаков. В сети BP-SOM, описанной в разделе 6.3, авторы работы [Weijters et al., 1997] использовали 12 входных элементов, чтобы представить месяцы

года. Разные элементы представляли разные месяцы. При наивном подходе можно было бы использовать один элемент и нумеровать месяцы от 1 до 12. Такая нумерация порождает смещение, в результате которого один месяц может иметь большее влияние, чем другой. Но в каком смысле декабрь может быть более важным, чем январь, с точки зрения вычисления данных?

6.4.4. Локальные минимумы

Для сети с обратным распространением ошибок не является редкостью попадание в локальный минимум. Локальный минимум может быть близким, но все же недостаточно близким к удовлетворительному решению проблемы. Метод градиентного спуска работает прекрасно, пока наклон функции ошибок все время ведет вниз, но на практике поверхность ошибок для большой сети будет иметь множество ложбин, холмов и складок. При таком рельефе будет достаточно мест, где можно застрять.

Проблему локальных минимумов может помочь преодолеть инерционный член, поскольку при его использовании изменения весовых значений включают часть предыдущего изменения. Если предыдущее изменение весового значения означало большой по величине шаг вниз по поверхности ошибок и сеть оказалась в локальном минимуме, то такое предыдущее изменение весового значения может помочь сети продолжить движение вниз, поскольку инерции окажется достаточно для того, чтобы преодолеть небольшую ложбину. Идея использования инерции иллюстрируется на рис. 6.19.

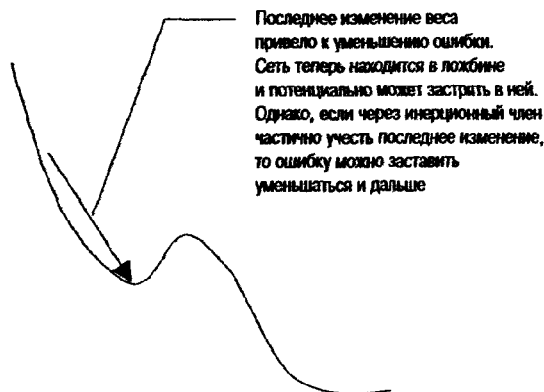


Рис. 6.19. Попытка обойти локальный минимум с помощью инерции

Если сеть ведет себя так, как будто она застряла в локальном минимуме, одним из простых решений является прекращение текущего процесса обучения и запуск его снова с новыми случайными весовыми значениями. Для больших сетей, когда для обучения сети до приемлемого уровня качества выполнения задачи требуются многочасовые усилия, необходимость начинать все сначала выглядит весьма непривлекательной. Иногда отойти от неудачного положения локального минимума на поверхности ошибок можно с помощью небольших случайных изменений всех весовых значений сети. Например, можно найти минимальное и максимальное весовые значения и к каждому весу добавить случайный процент (обычно меньше 10%) длины полученного таким образом диапазона.

Альтернативой указанной возможности обойти локальные минимумы в сетях с обратным распространением ошибок является использование метода модельной “закалки”. Как только сеть начинает проявлять признаки попадания в локальный минимум, используется метод модельной “закалки”, чтобы вернуться на спускающуюся вниз часть поверхности ошибок. Вернувшись на спускающийся участок, можно продолжать использование алгоритма обратного распространения ошибок.

6.4.5. Обобщение

Нейронная сеть оказывается совершенно бесполезной, если она не умеет обобщать полученную ею информацию. Обобщение означает способность сети качественно выполнять свою работу с данными, которые сеть в процессе обучения не видела. В случае сетей с обратным распространением ошибок плохие возможности обобщения могут быть следствием наличия слишком большого числа скрытых элементов или перетренировки сети.

Слишком много скрытых элементов могут оказаться препятствием для обобщения. Если сеть имеет слишком много ресурсов, то она может просто запомнить все учебное множество. Например, рассмотрим крайний случай, когда число скрытых элементов в сети с прямой связью соответствует числу учебных образцов. С таким объемом ресурсов сеть может запомнить весь учебный набор данных, посвятив по одному скрытому элементу каждому образцу. Но качество работы с данными, которые сеть еще не видела, скорее всего окажется весьма низким. Иногда можно избежать такого запоминания путем постоянного изменения учебных образцов, добавляя к образцу до 10% шума (точнее, случайную долю указанной величины) каждый раз при предъявлении образца сети. Шум должен добавляться к оригинальному чистому вектору, иначе со временем исходный учебный вектор может перестать быть характерным для решаемой проблемы.

Перетренировка при условии слишком большого числа скрытых элементов означает, что сеть повторяет учебные данные слишком хорошо и не позволяет новым экземплярам достаточно далеко отклоняться от учебных данных. Чтобы не допустить перетренировки, обучение можно периодически прерывать для того, чтобы пропустить через сеть тестовые данные. Если общая ошибка для тестового набора данных уменьшается, то обучение можно продолжить. Как только ошибка на тестовом множестве начинает расти, обучение прекращается. Контролируя процесс обучения с помощью тестового набора данных, можно предотвратить слишком близкую аппроксимацию учебного множества даже при наличии очень большого числа скрытых элементов. Если для контроля процесса обучения используется тестовое множество, необходимо также иметь набор для проверки правильности данных, аналогичный любому тестовому набору, но состоящий из данных, которые в ходе обучения не должны подаваться на рассмотрение сети ни в какой форме (кроме как для контроля).

6.5. Резюме

В последнее время наблюдается значительный рост интереса к использованию стохастических методов. Ряд моделей сетей с целью оптимизации использует идею модельной “закалки”. Метод модельной “закалки” может использоваться и для того, чтобы помочь алгоритмам градиентного спуска избежать попадания в локальные минимумы.

Вероятностная нейронная сеть (сеть PNN) в своей базовой форме может использоваться для классификации образцов. Она обладает следующими свойствами.

- Сеть PNN является моделью с управляемым обучением.
- Классификация неизвестных образцов основана на оценке вероятности того, что данный образец взят из имеющегося класса.
- Оценка вероятности вычисляется на основе распределения вероятностей учебных образцов.
- Обучение означает настройку параметров сети (весовых значений) на основе использования учебных экземпляров и экспериментирования с целью уточнения вероятностных оценок.

Доступность и качество учебных данных в конечном счете самым непосредственным образом влияют на качество работы нейронной сети. Данные нередко требуют предварительной обработки перед тем, как представить их на рассмотрение нейронной сети. Если нейронная сеть не может обобщать свои возможности на неизвестные ей данные, такая сеть не представляет никакой ценности.

6.6. Дополнительная литература

Обсуждение метода модельной “закалки” и машины Больцмана можно найти в большинстве книг по нейронным сетям. Но найдется совсем немного книг, в которых рассматривается вероятностная нейронная сеть. В [Masters, 1995] дается достаточно глубокий и вместе с тем полезный с практической точки зрения анализ вероятностной нейронной сети. От читателя требуются базовые знания языка C++ (или, по крайней мере, C).

6.7. Упражнения

1. Масштабируйте следующие данные к диапазону от 0.1 до 0.9:

–12, –8, –6, –2, 4, 8, 9, 15, 15.

2. Повторите вычисления упражнения 1 для диапазона от –0.9 до 0.9.
3. Большая торговая сеть по продаже автомобилей может предсказать тип автомобиля, который выберет потенциальный покупатель, основываясь на таких признаках, как уровень дохода, семейное положение и т.д. Признаки автомобиля должны включать его технические характеристики и дизайн. Как можно было бы кодировать признаки, такие как, например, цвет автомобиля? Составьте список признаков, которые, на ваш взгляд, можно было бы использовать для решения указанной задачи. В соответствии с этим списком сделайте оценку числа элементов, которые требуются для сети с прямой связью, чтобы выполнить поставленную задачу, а по размерам сети оцените размеры необходимого учебного множества. Обоснуйте свои ответы.
4. Нейронная сеть PNN обучается на строчных буквах алфавита. Для учебных данных используются шесть разных шрифтов. Предположим, что символы представляются в виде сетки из 11×11 точек. Укажите число элементов в такой сети PNN.
5. Повторите вычисления примера 6.2 для $\sigma = 0.3$.
6. Повторите вычисления примера 6.2 для $\sigma = 0.5$.
7. Повторите вычисления примера 6.2, но на этот раз используйте ненормализованные данные и метрику евклидова расстояния. Используйте $\sigma = 0.9$.
8. Повторите вычисления упражнения 7, используя $\sigma = 1.5$.
9. На рис. 6.20 показаны три класса данных и неизвестный экземпляр данных. Соответствующие точкам данные приведены в табл. 6.6. Вычислите класс, к которому принадлежит неизвестный образец, используя:

- (а) “ближайшего соседа”;
- (б) центроид;
- (в) сеть PNN с $\sigma = 0.9$.

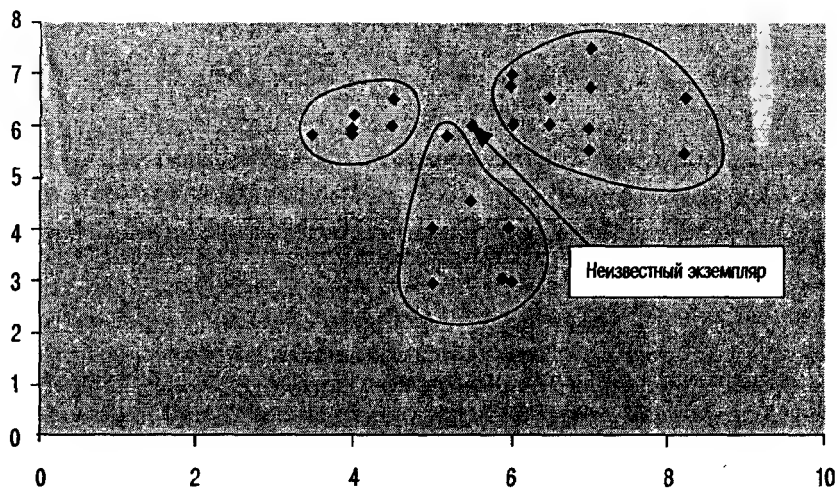


Рис. 6.20. Изображение данных для упражнения 9

Таблица 6.6. Данные, используемые в упражнении 9

Класс	x	y
Не известен	5.50	6.00
A	4.00	6.20
A	3.50	5.80
A	4.00	6.00
A	4.00	5.80
A	4.50	6.00
A	4.50	6.50
B	6.00	6.00
B	7.00	6.70
B	6.00	7.00
B	7.00	7.50

Класс	x	y
B	6.00	6.80
B	6.50	6.00
B	6.50	6.50
B	7.00	5.50
B	7.00	6.00
B	8.20	6.50
B	8.20	5.50
C	5.20	5.80
C	5.00	4.00
C	5.50	4.50
C	6.00	4.00
C	5.00	3.00
C	5.90	3.00
C	5.00	3.00
C	6.00	3.00

10. Повторите с помощью PNN классификацию из упражнения 9, используя $\sigma = 0.8$.
11. Повторите с помощью PNN классификацию из упражнения 9, используя $\sigma = 0.5$.

Глава 7

Связь с искусственным интеллектом

Задача. Введение в символьную парадигму искусственного интеллекта и связь с нейронными сетями.

Цели. Вы должны понять:

каково сегодняшнее состояние и цели изучения искусственного интеллекта;
главные составляющие интеллектуальных систем;
основные принципы символических рассуждений;
суть основных проблем, возникающих при разработке систем понимания речи.

Требования. Общие знания, касающиеся применения компьютерной техники.

7.1. Введение

Нейронные сети могут применяться при решении самых разных технических проблем, и мы уже видели несколько примеров такого их применения в предыдущих главах. Архитектура искусственных нейронных сетей не идет ни в какое сравнение с человеческим мозгом по сложности и размерам, но именно такие сети составляют абстрактные модели мозга. Поэтому сразу же возникает вопрос: можно ли с помощью искусственных сетей реализовать “искусственный интеллект”? Искусственный интеллект является одной из важнейших дисциплин в области информатики, имеющей глубокие связи с математикой, психологией и даже философией. Эта дисциплина была признана самостоятельной в 1950-х, и с тех пор на исследования по построению интеллектуальных машин было потрачено немало времени и денег. Цель изучения искусственного

интеллекта в широком смысле может быть описана как создание машин, выполняющих задачи, с которыми хорошо справляется человек, но которые нелегко запрограммировать с использованием традиционного подхода к вычислениям. Задачи, которые люди выполняют хорошо, но которые трудны для программирования, часто оказываются вполне обыденными. Например, хождение по комнате или поддержание разговора не требует таких умственных усилий, как сложение чисел. Но создание машин, выполняющих подобные задания, оказалось для инженеров проблемой огромной сложности. Тот факт, что относительно просто построить машину, способную сдать университетский экзамен, но не машину, способную выполнить уборку дома, делает вызов нашим традиционным представлениям о том, что значит быть интеллектуалом.

Исследователи искусственного интеллекта сосредоточились главным образом на том, что обычно называют *символьной парадигмой*, а также — “традиционным искусственным интеллектом” или “классическим искусственным интеллектом” (а иногда и “старомодным искусственным интеллектом”). Теперь существует новая форма искусственного интеллекта, основанная на коннекциях (т.е. нейронных сетях) и эволюционных моделях (т.е. системах, существующих в некоторой среде и адаптирующихся к ней на основе *естественного отбора*). Новый искусственный интеллект не отрицает традиционный искусственный интеллект, и некоторые считают, что не будет отрицать никогда. На самом деле для создания истинно интеллектуальных машин может потребоваться радикальный переворот в наших представлениях о них. Можно возразить, что новый искусственный интеллект является только новой формой традиционного искусственного интеллекта, и основанием для такого возражения является тот факт, что до сих пор на низшем уровне мы используем все те же вычислительные устройства (символьные системы на самом деле реализуются на последовательных машинах, то же самое можно сказать и о большинстве моделей нейронных сетей). Искусственный интеллект все еще обещает то, что обещалось еще в начале его развития, но пока что о важности искусственного интеллекта мы должны говорить с оптимизмом. В конце концов, наши запросы к качеству работы технических устройств все время растут, поэтому на любом этапе разработки таких устройств всегда неявно ставится задача создания машины с более высоким уровнем интеллекта. Несомненно, будут создаваться все более и более интеллектуальные машины, но мы всегда будем живо интересоваться возможностью создания андроида, способного вести с нами настоящий диалог. И даже когда такой андроид будет построен, философы все равно не перестанут спорить о том, обладает ли этот андроид интеллектом, сравнимым с нашим.

Чтобы развивать новый искусственный интеллект, нам нужно понять кое-что из традиционного искусственного интеллекта и, возможно, свести обе эти парадигмы вместе “под знаменем” некоторого общего искусственного интеллекта. В этой главе мы обсудим природу интеллекта и разработки в области традиционного искусственного интеллекта, чтобы в следующей главе обратиться к достижениям разработчиков нейронных сетей.

7.2. Природа интеллекта

Интеллект представляет собой концепцию, не поддающуюся строгому определению. Человеческий интеллект складывается из множества компонентов, среди которых способность к обучению, органы чувств (например, зрение и осязание), позволяющие взаимодействовать с внешней средой, и объем знаний, который вообще не поддается оценке. Даже наши домашние животные могут считаться интеллектуальными: они имеют сложные системы обработки информации, которые дают им возможность распознавать людей и выполнять задания, требующие немалого мастерства.

Интеллектуальное поведение не является исключительной прерогативой действующего в изоляции центрального процессора (человеческого мозга) — работа центрального процессора зависит от дополнительных приборов, воспринимающих, предварительно обрабатывающих и структурирующих поступающие данные. Интеллектуальные агенты могут быть построены таким образом, что им придется действовать в полностью виртуальном мире, но этим агентам необходимо будет замечать и развивать желания и цели, чтобы проявлять себя интеллектуально. Исследования обычно концентрируются на каком-то отдельном и узком аспекте интеллекта, и из практических соображений сенсорные каналы и первичная обработка данных часто заменяются предположением о наличии подходящей структуры данных. Традиционный искусственный интеллект заставлял нас верить, что теоретически возможно (но практически все еще нет) создать мозг — воплощение искусственного интеллекта — следует лишь создать правильный алгоритм, и мозг тут как тут. В рамках традиционного искусственного интеллекта изучение интеллектуальных систем концентрировалось вокруг нескольких ключевых вопросов — это представление данных, способность рассуждать и способность системы автоматически адаптироваться к изменению условий (другими словами, способность обучаться).

7.2.1. Знание и представление

Интеллектуальная система воплощает знание. Когда мы говорим о знаниях, это могут быть фактические знания, необходимые, например, для участия в викторине, или практические знания, применяемые при

замене автомобильного колеса. Это могут быть знания, называемые навыками, необходимые, например, при езде на велосипеде, или же это могут быть миллионы битов информации, которую мы называем здравым смыслом, позволяющим, например, не пролить воду из чашки при питье. Представление знания может быть явным или неявным.

Явное знание может быть установлено и инспектировано, например, в форме фактов.

Яблоко есть плод.

Кот есть животное.

Это могут быть также и правила.

Если аккумулятор разрядился, то автомобиль не заведется.

Если процентные ставки растут, то цена кредита поднимается.

Неявное знание передать непросто. Например, ребенку можно разъяснить общие принципы езды на велосипеде, включая необходимость вращать педали и направлять переднее колесо в сторону движения, но процедура не может быть записана так, чтобы ребенок мог ее прочесть, а затем использовать для того, чтобы поехать на велосипеде. В действительности знание того, как ехать на велосипеде, приобретается в результате проб и ошибок.

Традиционный искусственный интеллект складывается из обработки символов, а знания представляются с помощью символьных структур. Представление знания имеет несколько форм и различных уровней. Вот примеры представлений: знания, выраженные в виде правил; график изменения роста ребенка; карта лондонского метро. Уровень представления зависит от уровня детализации информации, которую необходимо передать. Например, карта лондонского метро представляет собой абстрактное изображение истинной сети связанных станций. Для целей планирования переездов между станциями карта оказывается вполне подходящей, так как показывает все соединения и взаимное расположение станций. Изображенные на карте линии не отражают реального расстояния и кривизну истинных линий, связывающих станции, так как эта информация к делу не относится, а только мешает зрительному восприятию.

Исследователи искусственного интеллекта используют знания о мозге как руководство для того, чтобы строить искусственный интеллект, но теорию того, как человек представляет знания, формализовать очень тяжело. Как люди воспринимают корешки книг — как канцелярские принадлежности, украшения или элементы обстановки? Как можно представить любовь, если машина не испытывает таких эмоций? Можно ли заставить машину испытать любовь и, если это возможно, то будет ли ее

понимание любви отличаться от нашего? Существуют когнитивные теории того, как представляет знания мозг человека, и именно такие теории служат основой для символических представлений.

7.2.2. Рассуждения

Человек знает, как вести себя в новых ситуациях, поскольку он может анализировать свои знания и принимать адекватные решения, основанные на прошлом опыте и на прогнозировании возможных последствий таких решений. Символическая парадигма пытается отчасти имитировать такое поведение с помощью программ, включающих набор правил представления соответствующих умозаключений.

Правило.	Если аккумулятор разрядился, то автомобиль не заведется.
Конкретные данные.	Аккумулятор в автомобиле Сюзен разрядился.
Новое выведенное знание.	Автомобиль Сюзен не заведется.

Совсем не сложно запрограммировать систему, работающую в соответствии с основными правилами логических заключений, но прежде, чем машина сможет демонстрировать нечто подобное реальному интеллекту, придется найти ответы на некоторые важные вопросы. Правила воплощают знания, а поэтому и вопросы оказываются связанными со знаниями. Какие правила требуются и являются достаточными для того, чтобы выполнить задачу интеллектуально? Рассмотрим специальную задачу, требующую только знания устройства автомобиля. Экспертная система (система, основанная на использовании знаний, необходимых для вывода заключений в узкой области) вовлекается в следующий диалог.

Клиент: Зачем на автомобилях устанавливаются ручные тормоза?

Экспертная система: Чтобы остановить движение автомобиля.

Клиент: Автомобиль ведь не всегда движется, когда ручной тормоз выключен?

Экспертная система: Это зависит от горизонтальности поверхности.

Клиент: Что дает энергию, заставляющую колеса автомобиля вращаться?

Экспертная система: Двигатель.

Клиент: Таким образом, если двигатель работает, то колеса будут вращаться?

Экспертная система: Если сцепление включено.

Клиент: Что случится, если двигатель будет работать со включенным ручным тормозом и включенным сцеплением?

Экспертная система: Предполагается, что вы сначала должны отключить ручной тормоз.

Клиент: Да, но предположим, что я этого не сделал.

Экспертная система: ???

Ответ “Не знаю” на последний вопрос является приемлемым, если вы не эксперт по автомобилям, но выглядит смешным, если предполагается, что вы эксперт. Тот, кто никогда не водил автомобиль, но имеет хорошее знание физики и механических устройств, вполне может предложить разумный ответ. Но знание физики требует выхода экспертной системы из узкой области автомобилей и ведет к необходимости освоения более широкой области знаний. Понимание того, сколько требуется знаний для адекватного выполнения соответствующей задачи, является проблемой разработчика экспертной системы: достаточно трудно гарантировать правильную работу компьютерной программы, допускающую ограниченное число типов ввода, но трудность проверки правильности работы возрастает многократно, когда на этапе проектирования потенциальный ввод не может быть предсказан вообще. Существуют впечатляющие экспертные системы, выполняющие огромную работу и, как сообщается, значительно уменьшающие расходы компаний. Но потребуйте от экспертной системы слишком много — и она вскоре покажется вам слабой, поскольку знания ее ограничены.

У специалистов, создающих экспертные системы, возникают и другие трудности. Рассмотрим утверждение, подобное следующему.

Чувство было просто ужасным, это счастье остаться после такого в живых, ведь все мужчины в самолете погибли.

Является ли рассказчик женщиной? Находилась ли она в самолете?

Я не сомневаюсь, что они возложат всю вину на женщину-пилота.

Теперь кажется разумным предположить, что рассказчиком является женщина-пилот и что она была единственной женщиной на борту самолета, потерпевшего аварию.

Мы столкнулись на подходе к аэродрому. Я была в одноместном самолете, и у меня было катапультируемое кресло.

Последняя информация заставляет нас пересмотреть предыдущие выводы. Построение интеллектуального агента, способного делать серии заключений, но исправляющих эти заключения на последующих стадиях, является только одной из множества проблем, стоящих перед разработчиками интеллектуальных систем.

7.2.3. Обучение

Выживание любого животного зависит от способности адаптироваться к изменению ситуации на основе имеющегося опыта. Мы полагаемся на то, что наши домашние животные способны к обучению в такой мере, что мы сможем научить их вести себя социально приемлемым образом.

Большинство компьютеров программируются для выполнения определенных задач. Инженер-программист представляет описание проблемы в виде алгоритма, а затем реализует этот алгоритм на каком-нибудь языке программирования высокого уровня. Связь проблемы с реализацией опирается на знание и представление. Например, можно заявить, что площадь круга равна πr^2 , где π является числовой константой, а r — радиусом. Понимание этого описания опирается на знание того, что радиусом является длина отрезка прямой, соединяющего центр круга с точкой на окружности, что такое окружность и т.д. Предложенная выше формула выражает знание, но использование этой формулы тоже требует знаний: например, пользователь должен знать, что площадь будет выражаться в единицах, представляющих собой квадрат единиц, использующихся для измерения радиуса.

Вот пример программы на языке C для вычисления площади круга по указанной формуле:

```
float areaOfCircle(float radius)
{
    float pi = 3.1415927;
    return(pi*radius*radius);
}
```

Внутренне компьютер представляет число π в двоичном формате, но это не имеет никакого значения для программиста, равно как и реализация именно на языке C не имеет никакого значения для пользователя программы, поскольку понимание данного алгоритма зависит только от достаточно хорошего знания английского языка и некоторых математических результатов.

Ключом к программированию является способность выразить проблему в виде алгоритма и выбрать подходящее представление. Но что можно сказать о знании, которое не может быть выражено явно? Невозможность явно выразить решение означает, что программа не может быть получена традиционными методами. Но можно создать программу автоматически в результате процесса обучения. Машины могут программироваться так же, как происходит обучение собаки выполнению команды. Команда служит вводом, правильный ответ вознаграждается, и сигнал повторяется до тех пор, пока собака не адаптирует свое поведение в соответствии с требуемым ответом.

Имеется много символьных алгоритмов обучения, но объем книги не предусматривает их описания в этой главе. Заинтересовавшиеся читатели могут ознакомиться с дополнительной литературой по этому вопросу, указанной в разделе 7.8.

7.3. Гипотеза символьных систем

Традиционный искусственный интеллект опирается на гипотезу символьных систем. Эта гипотеза в сильно упрощенной формулировке утверждает, что, реализовав огромную структуру взаимосвязанных символов, представляющую реальные знания, и сложный набор символьных процессов, позволяющих оперировать структурами с целью создания новых структур, можно создать машину, работающую так же хорошо, как человек. Коротко говоря, гипотеза утверждает, что можно запрограммировать компьютер так, что он будет думать.

В этом разделе мы рассмотрим пару простых модельных задач, позволяющих понять, что такое символьная обработка. Мы не будем обсуждать здесь правильность указанной гипотезы, поскольку хотели бы только выяснить, что такое символьные процессы. Необходимо также иметь в виду, что если на базе этой гипотезы мыслящая машина может быть построена, то такая машина должна концентрировать колоссальные объемы знаний, объединенных в огромный массив сложных процессов. В этой главе мы рассмотрим природу вычислений. По практическим соображениям почти все эксперименты с искусственным интеллектом имели малые масштабы, но если создание машины, подобной человеку, возможно, масштабы этих экспериментов должны существенно измениться. Движение в этом направлении уже началось (см. раздел 7.8).

7.3.1. Поиск

Решение головоломок, построение планов, диагностика заболеваний и т.д. — все это проблемы, которые можно решить в результате процесса поиска. Рассмотрим планирование автобусного путешествия по Англии с юга на север. В вашем распоряжении имеется расписание движения автобусов, а задачей является нахождение последовательности стыкующихся маршрутов, которые позволят вам добраться из Саутгемптона в Ноттингем (предполагается, что прямого сообщения между этими городами нет). Можно начать с Саутгемптона, как пункта отправления, просмотреть список пунктов назначения маршрутов, выбрать некоторый пункт назначения и перейти на ту страницу расписания движения, которая соответствует выбранному пункту назначения. Пункт назначения теперь

становится вашим воображаемым пунктом отправления, и вы выбираете другой пункт назначения. Вы продолжаете эту процедуру до тех пор, пока не увидите название пункта отправления, для которого пунктом назначения будет указан Ноттингем. Нет ничего проще! На практике, однако, вы воспользуетесь скорее всего имеющимися у вас знаниями географии и будете рассматривать только те промежуточные пункты назначения, которые лежат в направлении, приближающем вас к конечному пункту назначения, Ноттингему. Вы можете также использовать несколько закладок, чтобы, найдя некоторый маршрут из Саутгемптона в пункт А, перейти к соответствующей Ноттингему странице расписания и посмотреть, есть ли маршрут из Ноттингема в пункт А. И вы будете продолжать перескакивать так с одной страницы на другую в надежде найти пункт, где ваш маршрут вперед и обратный маршрут пересекутся.

Были разработаны компьютерные алгоритмы, имитирующие такой тип поиска. Давайте рассмотрим одну простую головоломку, подобную “игре в пятнадцать”, но из восьми элементов. Головоломка представляет собой решетку с девятью ячейками, где восемь ячеек заняты пронумерованными плитками. Незанятую ячейку можно интерпретировать как пустую плитку: соответствующей плитки нет, но это оказывается как полезной идеей, позволяющей упростить описание проблемы. Плитки размещаются случайным образом, а целью является размещение плиток в порядке номеров слева направо и сверху вниз. Задача ставится скучная и несколько трудоемкая: вы завязываете другу глаза и даете ему головоломку для того, чтобы он передвигал плитки. Ваш друг передвигает одну плитку, а вы проверяете результат. Плитки не размещаются в нужном порядке, поэтому вы просите друга передвинуть еще одну плитку. Какую плитку двигать, решает ваш друг, а не вы. После очередного передвижения вы выполняете очередную проверку, и этот процесс продолжается до тех пор, пока плитки не будут размещены в надлежащем порядке.

Плитки перемещаются снова в случайном порядке. На этот раз вы ведете запись выполненных перемещений, чтобы ваш друг не повторял одни и те же последовательности перемещений из одних и тех же комбинаций плиток. Ваш друг выполняет поиск вслепую.

На каждой стадии поиска для перемещения существует определенное число возможностей. Чтобы упростить описание, представим себе, что передвигается пустая плитка и что она может двигаться вверх, вниз, вправо или влево, в зависимости от того, в каком месте решетки она находится. Единственной формой помощи, которую вам разрешается оказать вашему другу, является сигнал завершения выполнения задания и недопущение повторения последовательности перемещений. Каждая комбинация плиток

называется *состоянием*, так что задачей является нахождение целевого состояния — того, в котором все плитки оказываются в нужном порядке. Можно рассмотреть случайное начальное состояние и выяснить, какие новые состояния могут быть из него получены с помощью перемещения пустой плитки. Из каждого нового состояния процесс можно продолжить дальше. Соответствующая идея представлена на рис. 7.1. Показаны не все, а только некоторые состояния. От каждого состояния отходит несколько линий, каждая из которых соответствует определенному перемещению пустой плитки. В результате такого перемещения возникает новое состояние решетки. Если продолжить построение иллюстрации дальше, мы увидим, что в нескольких местах иллюстрации будут размещены состояния, соответствующие конечной цели. В результате движения по маршруту перемещений от начального состояния (корня) к любому из целевых состояний выявляется последовательность перемещений, дающая решение. Одни последовательности могут при этом оказаться короче других.

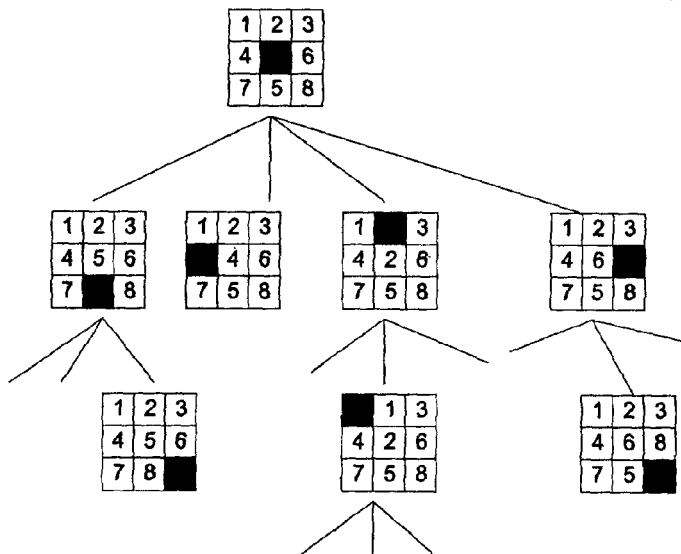


Рис. 7.1. Некоторые из состояний поиска, генерируемые при попытках решить головоломку из восьми плиток. Целевое состояние находится в нижнем ряду слева. Для начального состояния перемещение пустого квадрата вверх эквивалентно перемещению "двойки" вниз, но описание головоломки в терминах движения пустого квадрата оказывается более удобным

Алгоритмы поиска вслепую легко реализуются на компьютере. Трудности обычно заключаются в выяснении того, от чего зависят состояния и как лучше всего эти состояния представить. Для головоломки из восьми плиток представление состояний оказывается простым, но для реальных задач найти подходящее представление бывает довольно непросто.

Если вы сами решаете головоломку из восьми плиток, то, вероятно, найдете решение быстрее, чем это сделал бы ваш друг с завязанными глазами. Вы будете делать для себя выводы о том, приближает ли каждое конкретное перемещение вас к конечной цели. Этот информирующий вариант игры называется *эвристическим поиском*. Поиск подходящего маршрута по расписанию с использованием знания географии является эвристическим: для направления поиска используются подходящие знания. Разработка алгоритмов для осуществления эвристического поиска не намного сложнее программирования поиска вслепую. Самым сложным моментом здесь является определение эвристик, т.е. правил поиска.

Поиск является основой искусственного интеллекта. Как поиск может быть представлена и диагностика состояния здоровья пациента с помощью применения правил, и некоторые аспекты проблемы понимания речи.

7.3.2. Продукционные системы

Продукционная система несет в себе фундаментальные идеи, на которых “выросли” экспертные системы и языки типа Prolog. “Сердцем” продукционной системы является процедура управления, остающаяся неизменной во всех приложениях. Вычисления, которые выполняет продукционная система, управляются набором продукционных правил и конкретными данными. Продукционные правила образуют множество пар “условие—действие”, определяющих условия, которые должны быть выполнены для того, чтобы правило оказалось разрешенным к применению, и действие, которое должно быть выполнено, если это правило применить. *Рабочая память* хранит текущее состояние, и именно от текущего состояния зависит, будет ли выполняться условие конкретного правила. Продукционный цикл исключительно прост: по состоянию рабочей памяти следует сначала проверить, какие правила могут быть удовлетворены, сделать разрешенными к применению все удовлетворенные правила, из множества разрешенных к применению правил выбрать правило, которое следует применить, и обновить состояние рабочей памяти соответственно действию примененного правила.

Множество разрешенных к применению правил называется *множеством противоречий*, а стратегия выбора из этого множества правила, которое следует применить, называется *стратегией разрешения противоречий*.

Весь процесс выбора правила для применения повторяется до тех пор, пока не будет удовлетворено некоторое условие остановки. Простой пример такого процесса показан на рис. 7.2. В этом примере стратегия разрешения противоречий представляет собой стратегию *новизны*, означающую просто выбор правила, условие которого ранее еще не было удовлетворено, а в случае нескольких таких правил — произвольный выбор любого из них. Определение стратегии разрешения противоречий является задачей разработчика. Существует несколько стандартных стратегий, например уже упоминавшаяся стратегия новизны или стратегия конкретности (когда преимущество имеет правило, имеющее большее число условий), но читатель должен понять, что детали конструкции любой системы оставляются на усмотрение ее разработчика: продукционная система является общей вычислительной процедурой, которую можно настраивать так, как этого требует конкретная задача. Пример последовательности действий, выполняемых продукционной системой, показан на рис. 7.3.

Продукционное множество:

- 1 $P \wedge Q \wedge R \Rightarrow \text{цель}$
- 2 $S \Rightarrow R$
- 3 $T \wedge W \Rightarrow Q$
- 4 $T \Rightarrow S$

Цикл	Рабочая память	Множество противоречий	Выбранное правило
0	T, W, P	3, 4	3
1	T, W, P, Q	3, 4	4
2	T, W, P, Q, S	2, 3, 4	2
3	T, W, P, Q, S, R	1, 2, 3, 4	1
4	T, W, P, Q, S, R, цель	1, 2, 3, 4	Останов

Рис. 7.2. Шаги продукционной системы при выводе целевого заключения (T, W и P существуют, как факты, с самого начала)

7.4. Представление с помощью символов

Чтобы строить интеллектуальные системы, язык представлений должен отвечать следующим требованиям.

- Быть достаточно выразительным, чтобы позволять представление всех реальных состояний.

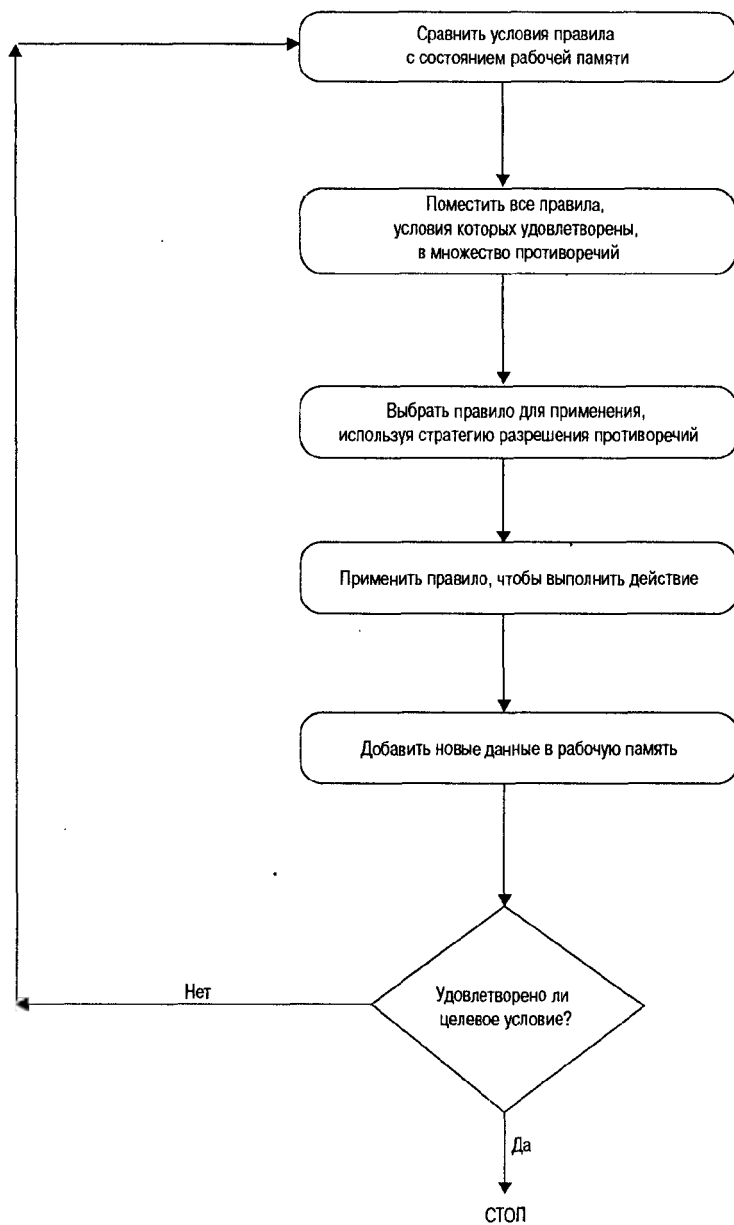


Рис. 7.3. Цикл продукционной системы

- Быть сжатым настолько, чтобы вычисления поддавались управлению.
- Иметь форму, подходящую для рассуждений, чтобы модель реальности могла менять свое состояние.

Логика используется математиками для того, чтобы доказывать правильность или неправильность утверждений. Все главные языки представлений, включая Rules, Semantic nets и Frames, могут быть переведены в форму логики, известную как исчисление предикатов первого порядка. Краткое обсуждение исчисления предикатов первого порядка дается сразу после рассмотрения более простой формы логики, называемой пропозиционным исчислением или исчислением высказываний.

7.4.1. Исчисление высказываний

Высказыванием является предложение, которому может быть присвоено значение ИСТИНА или ЛОЖЬ. Например, “Сегодня идет дождь” или “Медь есть металл” являются или правильными, или ложными утверждениями. В рамках исчисления высказываний можно представлять предложения в символьном виде, создавать сложные предложения из других предложений, используя связки, и выводить заключения, чтобы найти значение предложения. Синтаксис исчисления высказываний описывает то, как создаются предложения (включая сложные выражения), грамматика определяет правила, по которым выясняется синтаксическая корректность предложения. Семантика исчисления высказываний определяет то, как определить соотношение между предложением и значениями ИСТИНА или ЛОЖЬ.

Элементарным предложением является простое высказывание типа “Сегодня идет дождь”, и такие предложения часто представляются прописными буквами, например P или Q. Сложное предложение создается с помощью связки, соединяющей два или несколько элементарных предложений. Круглые скобки используются для того, чтобы указать приоритет операций при комбинировании элементарных предложений с помощью связок. В табл. 7.1 дано объяснение связок, а в табл. 7.2 приведены правила грамматики для логики высказываний. Грамматика может использоваться для анализа предложения и проверки того, что оно синтаксически правильно. Например, неформальное описание проверки синтаксической правильности выражения $((P \wedge Q) \wedge R) \Rightarrow S$ предлагается в табл. 7.3.

Таблица 7.1. Логические связи

Связка	Пример
$\wedge$ (И), называемая конъюнкцией	Сегодня идет дождь И я промок $P \wedge Q$, где P обозначает конъюнкт "Сегодня идет дождь", а Q — конъюнкт "Я промок"
$\vee$ (ИЛИ), называемая дизъюнкцией	'Салли пошла в магазин ИЛИ Том пошел в магазин $P \vee Q$, где P обозначает дизъюнкт "Салли пошла в магазин", а Q — дизъюнкт "Том пошел в магазин"
$\neg$ (НЕ), называемая отрицанием	Слон НЕ розовый. $\neg P$, где P обозначает "Слон розовый"
$\Rightarrow$ (ВЛЕЧЕТ), называемая импликацией или утверждением "если ..., то ..."	ЕСЛИ я на улице И идет дождь, ТО я промокну $(P \wedge Q) \Rightarrow R$, где P обозначает "Я на улице", Q обозначает "Идет дождь", а R — "Я промокну"
$\Leftrightarrow$ (ЭКВИВАЛЕНТ), называемая эквивалентностью или двойной импликацией	Выражения по обе стороны этой связи должны быть логически эквивалентными

Таблица 7.2. Правила грамматики для логики высказываний

Правило грамматики	Объяснение
Предложение $\rightarrow$ Элементарное – предложение Сложное – предложение	Предложение является либо элементарным, либо сложным
Элементарное – предложение $\rightarrow P Q R ...$	Элементарное предложение обозначается прописной буквой, как правило из последней трети алфавита
Элементарное – предложение $\rightarrow$ ИСТИНА ЛОЖЬ	Элементарное предложение может представляться также значениями ИСТИНА или ЛОЖЬ
Сложное – предложение $\rightarrow$ Предложение Связка Предложение $\neg$ Предложение (Предложение)	Сложное предложение строится из двух других предложений и связки. Отрицание предложения тоже является предложением, и помещение предложения в скобки тоже в результате дает предложение
Связка $\rightarrow \wedge \vee \neg \Rightarrow \Leftrightarrow$	Связкой является любой из логических символов, представленных в табл. 7.1

Таблица 7.3. Пример синтаксического анализа

Шаги	Объяснение
$((P \wedge Q) \wedge R) \Rightarrow S$	В соответствии с первым правилом предложение является или элементарным, или сложным. Очевидно, что это выражение не является элементарным предложением, поэтому мы должны проверить, что оно является сложным предложением
$((P \wedge Q) \wedge R) \Rightarrow S$ $\underbrace{\hspace{1.5cm}}$	Правой стороной выражения является S, а S является элементарным предложением в соответствии со вторым правилом
$((P \wedge Q) \wedge R)$	Мы должны показать, что $(P \wedge Q) \wedge R$ является сложным предложением
$(P \wedge Q) \wedge R$ $\underbrace{\hspace{1.5cm}}$	Сравнивая с правилом "Предложение Связка Предложение", делаем вывод, что R является предложением, поскольку оно является элементарным
$(P \wedge Q)$	Мы должны показать, что $P \wedge Q$ является сложным предложением
$P \wedge Q$ $\underbrace{\hspace{1.5cm}}$	Снова сравнивая с правилом "Предложение Связка Предложение", делаем вывод, что и P, и Q являются предложениями

Семантика исчисления высказываний определяется с помощью *таблицы истинности*. Все связки, за исключением отрицания, являются бинарными отношениями, поэтому в определениях требуется использовать два символа. Предложение может принимать значение ИСТИНА или ЛОЖЬ, поэтому для бинарных связок имеется четыре возможные комбинации, как показано в табл. 7.4.

Таблица 7.4. Таблица истинности

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
ИСТИНА	ИСТИНА	ЛОЖЬ	ИСТИНА	ИСТИНА	ИСТИНА	ИСТИНА
ИСТИНА	ЛОЖЬ	ЛОЖЬ	ЛОЖЬ	ИСТИНА	ЛОЖЬ	ЛОЖЬ
ЛОЖЬ	ИСТИНА	ИСТИНА	ЛОЖЬ	ИСТИНА	ИСТИНА	ЛОЖЬ
ЛОЖЬ	ЛОЖЬ	ИСТИНА	ЛОЖЬ	ЛОЖЬ	ИСТИНА	ИСТИНА

По большей части определения связок интуитивно понятны. Например, "Сегодня идет дождь" И "Сегодня НЕ идет дождь", очевидно, ЛОЖЬ, что подтверждается таблицей истинности, если найти в ней возможные значения для ИСТИНА $\wedge$ ЛОЖЬ или ЛОЖЬ $\wedge$ ИСТИНА. Опре-

деление импликации часто вызывает вопросы, поскольку мы интуитивно пытаемся применить интерпретацию, основанную на нашем общем понимании языка. Например, можно заявить, что предложение типа “Если число 5 четно, то мой автомобиль розовый” (имеющее форму $P \Rightarrow Q$) является ложью, поскольку оно не имеет смысла. Однако, в соответствии с определением импликации, это выражение оказывается истинным, вне зависимости от того, является ли ваш автомобиль розовым или нет: мы знаем, “число 5 четно” является ложью, так что соответствующей комбинацией должна быть либо ЛОЖЬ $\Rightarrow$ ИСТИНА (автомобиль розовый) или ЛОЖЬ $\Rightarrow$ ЛОЖЬ (автомобиль не розовый). Мы стремимся интерпретировать выражение типа “если ..., то ...” как причинно-следственное, и именно поэтому предложение “Если число 5 четно, то мой автомобиль розовый” выглядит бессмысленным, так как определение числа 5 не оказывает никакого влияния на цвет вашего автомобиля. Здесь следует понять, что импликация в исчислении выражений не требует, чтобы имелась хоть какая-то смысловая связь между предложениями по обе стороны связывающей их импликации. По импликации, знание того, что P является ложью, не дает никакой возможности сделать вывод об истинности Q . Например, предложение “Если аккумулятор разрядился, то автомобиль не заведется” дает нам возможность заключить, что истиной является “автомобиль не заведется”, если истиной является “аккумулятор разрядился”, но правило не говорит нам ничего о способности автомобиля завестись, если мы не знаем, что “аккумулятор не разрядился” — автомобиль может не завестись по целому ряду причин совершенно иного рода. Если этих объяснений вам все еще недостаточно, лучше всего интерпретировать $P \Rightarrow Q$ просто как предложение, позволяющее сделать вывод о том, что Q есть ИСТИНА, если мы знаем, что P есть ИСТИНА.

Существует целый ряд правил вывода логики высказываний, и они представлены в табл. 7.5.

Таблица 7.5. Правила вывода логики высказываний

Правило	Объяснение
Modus ponens: $\frac{A \Rightarrow B, A}{B}$	Если посылка (A) есть ИСТИНА, то и заключение (B) есть ИСТИНА
Исключение И: $\frac{A \wedge B}{A}, \frac{A \wedge B}{B}$	Знание того, что A И B есть ИСТИНА, должно означать, что A есть ИСТИНА и что B есть ИСТИНА

Правило	Объяснение
Интродукция ИЛИ: $\frac{A \quad B}{A \vee B, A \vee B}$	Если А есть ИСТИНА, то А ИЛИ В есть ИСТИНА; то же самое имеет место, если В есть ИСТИНА
Интродукция И: $\frac{A \quad B}{A \wedge B}$	Если А есть ИСТИНА и В есть ИСТИНА, то А И В есть ИСТИНА
Двойное отрицание: $\frac{\neg\neg A}{A}$	Если А есть НЕ НЕ ИСТИНА, то А есть ИСТИНА
Единичная резолюция: $\frac{A \vee B, \neg B}{A}, \frac{A \vee B, \neg A}{B}$	Если А ИЛИ В есть ИСТИНА и НЕ В, то А есть ИСТИНА. Точно так же если НЕ А, то В есть ИСТИНА
Резолюция: $\frac{A \vee B, \neg B \vee C}{A \vee C}$	Если А ИЛИ В и НЕ В ИЛИ С, то, поскольку В не может быть ИСТИНА и ЛОЖЬ одновременно, должно быть А ИЛИ В есть ИСТИНА

Пример 7.1

Имеется следующая информация.

Если аккумулятор машины разряжен, то машина не заводится. Если машина Джона не заводится и текущее время оказывается позже 8 часов утра, то Джон опоздает на поезд. Однажды утром после 8 часов утра аккумулятор машины Джона оказался разряженным.

Используя логические правила вывода, покажите, что Джон опоздал на поезд.

Решение 7.1

В символьном виде информация может быть представлена следующим образом.

P: аккумулятор машины разряжен.

Q: машина не заводится.

R: время после 8 утра.

S: Джон опоздал на поезд.

Правило 1. $P \Rightarrow Q$.

Правило 2. $Q \wedge R \Rightarrow S$.

Известно, что P и R есть ИСТИНА. Задачей является доказательство S . Доказательство строится следующим образом.

- 1 P Дано.
- 2 R Дано.
- 3 Q Следует из шага 1 и правила 1 по правилу *modus ponens*.
- 4 $Q \wedge R$ Следует из шагов 3 и 2 по правилу интродукции $\wedge$.
- 5 S Следует из шага 4 и правила 2 по правилу *modus ponens*.

7.4.2. Исчисление предикатов

Исчисление высказываний предполагает, что мир можно моделировать с помощью фактов. Но для реальных приложений исчисление высказываний не подходит. Рассмотрим, например, число предложений, которые потребуются для моделирования опоздания на работу, скажем, 50 человек. Для этого требуется язык, более выразительный в том смысле, что он должен допускать обобщенные выражения. Исчисление предикатов (или, более точно, исчисление предикатов первого порядка) расширяет язык исчисления высказываний так, что мир оказывается состоящим из объектов, отношений и свойств. Этот язык допускает обобщенные утверждения, вводя в рассмотрение переменные, а также кванторы, позволяющие определять свойства совокупности объектов. Рассмотрим утверждение “Если лошадью владеет Джон, то лошадь является чистокровной”. Это утверждение ссылается на совокупность объектов, а именно, на всех лошадей, которыми владеет Джон, вследствие чего опускается необходимость ссылаться на каждый экземпляр (т.е. на каждую из лошадей Джона) в отдельности. Это утверждение является обобщенным и избавляет от необходимости формулировки соответствующего утверждения для каждой из лошадей Джона. Можно перефразировать утверждение и сказать “Все лошади, которыми владеет Джон, являются чистокровными”. Здесь для указания того, что свойство “чистокровный” применимо ко всей совокупности лошадей Джона, используется квантор (все). Слово “владеет” является бинарным предикатом, описывающим отношение между Джоном и лошадью, а слово “чистокровный” является унарным предикатом, описывающим свойство лошади. В терминах исчисления предикатов утверждение “Все лошади, которыми владеет Джон, являются чистокровными” в символьном виде записывается так:

$$\forall x(\text{Лошадь}(x) \wedge \text{Владеть}(\text{Джон}, x)) \Rightarrow \text{Чистокровный}(x).$$

Символ $\forall$ называется *квантором общности* и читается как “для всех” или “каждый”. В исчислении предикатов определяется и другой символ, а именно, символ $\exists$, называемый *квантором существования*, который чита-

ется как “существует”. Например, когда мы говорим “Джон имеет чистокровную лошадь”, это означает, что существует по крайней мере одна лошадь (а возможно и больше) в коллекции Джона, являющаяся чистокровной. На языке предикатов это утверждение выглядит следующим образом:

$\exists x \text{ Лошадь}(x) \wedge \text{Владеть}(\text{Джон}, x) \wedge \text{Чистокровный}(x)$.

Исчисление предикатов предлагает язык более сжатый, чем обычный разговорный язык. В системе исчисления предикатов одно предложение будет соответствовать множеству предложений обычного разговорного языка. Исчисление предикатов уменьшает и неоднозначность. Например, “Все лошади не являются чистокровными” можно спутать с “Не все лошади являются чистокровными”, и путаница разрешается с помощью знания того, что существуют другие породы лошадей. Представление каждого из предложений в терминах предикатов показывает их разницу: $\forall \text{лошади} \neg \text{Чистокровный}(\text{лошадь})$ и $\neg \forall \text{лошади} \text{Чистокровный}(\text{лошадь})$. Первое предложение действительно утверждает, что все чистокровные вымерли (ни одного экземпляра не существует), а второе предложение утверждает, что “то, что каждая лошадь является чистокровной, неверно”.

7.4.3. Другие символьные языки

Есть много языков представлений, но они имеют много общего. Популярные языки типа FOPC, Frames или Semantic nets по существу эквивалентны и имеют целый ряд общих ключевых элементов:

- *объекты*, такие как, например, мяч, персона, лодка, велосипед или философ;
- *отношения* между объектами, такие как, например, Джон есть отец Ким, или паровоз тянет состав;
- *свойства*, такие как, например, моя машина зеленого цвета, Дэвид имеет рост шесть футов.

Эти языки на самом деле отличаются только синтаксисом. Синтаксис является важным, поскольку для выражения информации один формализм может оказаться более подходящим, чем другой. Например, легче найти маршрут между двумя станциями лондонского метро, используя графическую схему сети, чем описание соединений между ее станциями в виде текста. Выбираемый нами язык должен быть настолько выразительным, чтобы мы могли выразить все необходимые знания, но достаточно сжатым, чтобы можно было эффективно организовать необходимые вычисления. Кроме того, язык должен позволить выводить новые знания из уже имеющихся.

7.4.4. Язык Prolog

Язык Prolog формально не является языком представлений, но его синтаксис используется в следующей главе, поэтому мы приведем здесь его описание.

Язык Prolog является языком программирования искусственного интеллекта, в котором данные представляются как множество отношений между объектами. Программа на языке Prolog складывается из списка фактов и правил, и этот язык является привлекательным по причине того, что он имеет встроженный механизм вывода заключений, который дает возможность получить ответы на вопросы о запрограммированном знании. Синтаксис языка Prolog опирается на исчисление предикатов. Отношение типа “Джон любит Мери” записывается так:

любит(джон, мери).

Предикат (т.е. отношение) и объекты отношения должны начинаться с буквы нижнего регистра. Указанное выше отношение трактуется как факт, и другими примерами фактов являются:

металл(медь)

медь есть металл;

играют(джон, мери, теннис) Джон и Мери играют в теннис.

Переменная должна начинаться с символа верхнего регистра. Когда переменная ссылается на константу (т.е. на конкретный экземпляр, такой как джон), говорят, что переменная *конкретизирована*. Переменная может быть конкретизирована в результате сравнения выражений. Например, предположим, что база данных содержит любит(джон, мери). Тогда могут быть заданы следующие вопросы.

Кто любит мери?

любит(X, мери)

ответ X = джон

X конкретизирована значением джон

Кто является тем, кого любит джон?

любит(джон, X)

ответ X = мери

Кто кого любит?

любит(X, Y)

ответ X = джон, Y = мери

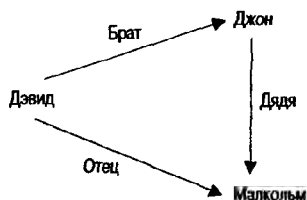
Правила выражаются с помощью обратной нотации в форме “условие если условие”, в противоположность прямой нотации “если условие, то условие”. Например,

дядя(X, Y) :- отец(Z, Y), брат(Z, X).

Символ “:-” используется вместо ЕСЛИ, а символ “,” — вместо И. Таким образом указанное правило утверждает, что

X есть дядя Y, если Z есть отец Y и Z есть брат X.

Пример конкретизации этого правила показан на рис. 7.4.



дядя(джон, малкольм) :- отец(дэвид, малкольм), брат(дэвид, джон).

Рис. 7.4. Правило языка Prolog, конкретизированное семантическим представлением, показанным в виде диаграммы

Переменная является локальной по отношению к выражению (т.е. факту или правилу): переменная с одним и тем же именем, появляющаяся в двух разных правилах, будет интерпретироваться в языке Prolog как имеющая разные имена.

Работа Prolog построена на унификации. Когда Prolog пытается доказать утверждение, ищется первое выражение, дающее совпадение. Для совпадения двух выражений они должны:

- иметь одно и то же отношение;
- иметь одинаковое число аргументов;
- давать совпадение по каждому аргументу.

Сравнение аргументов определяется следующим образом.

1. Константы: две константы дают совпадение, если они являются идентичными строками.
2. Константа и переменная: если переменная не конкретизирована, то она будет давать совпадение с любой константой и будет конкретизирована этой константой; конкретизированная переменная будет давать совпадение с константой в соответствии с п. 1.
3. Переменные: две произвольные переменные (не конкретизированные) всегда дают совпадение, а если в дальнейшем одна переменная становится конкретизированной некоторой константой, то другая переменная также будет конкретизирована той же константой.

Примеры совпадений показаны в табл. 7.6

Таблица 7.6. Сравнение образцов в языке Prolog

Выражение 1	Выражение 2	Совпадение
любит(Х, мери)	любит(джон, мери)	да
любит(джон, Х)	любит(джон, мери)	Х = мери
любит(джон, мери)	нравится(джон, мери)	нет
любит(Х, Y)	любит(джон, мери)	Х = джон Y = мери
любит(джон, мери)	любит(Х, Х)	нет
любит(Х, мери) и Х = джон	любит(Y, мери)	Y = Х = джон
толкает(джон, крис)	толкает(джон, крис, рука)	нет

Пример 7.2.

Ниже приведена программа на языке Prolog. Объясните, как ответит Prolog на запрос

?-дядя(джон, Х),

где “?-” является приглашением Prolog. Листинг программы следующий.

- 1 отец(дэвид, малкольм) .
- 2 дядя(грэхем, малкольм) .
- 3 брат(дэвид, джон) .
- 4 дядя(Х, Y) :- отец(Z, Y), брат(Z, X) .

Номера строк показаны для удобства.

Решение 7.2.

Prolog будет искать выражение с предикатом “дядя” и двумя аргументами. Первое такое выражение находится в строке 2, но совпадения не наблюдается, поскольку первый аргумент “грэхем” не соответствует аргументу “джон”. Второе выражение, содержащее “дядя”, находится в строке 4, но для того, чтобы Prolog “достиг успеха” в достижении цели (целью является показать, что Джон есть чей-то дядя), должна быть достигнута удовлетворена правая сторона выражения: должны быть совпадения для отец(Z, Y) и брат(Z, X). Х в запросе и Х в правиле считаются различными, так что мы имеем следующее:

Запрос: дядя(джон, Х1)

должен сравниваться с дядя(Х2, Y), что является левой частью правила в строке 4. Совпадение получится для Х2 = джон и Y = Х1

(или $X1 = Y$). Теперь задачей является доказать отец($Z, X1$) и брат($Z, джон$). отец($Z, X1$) даст совпадение со строкой 1 при $Z = дэвид$ и $X1 = малкольм$. Теперь задачей является доказать брат($дэвид, джон$), что удовлетворяется строкой 3. Prolog ответит $X = малкольм$.

7.5. Понимание речи

Для людей общение с помощью обычного разговорного языка является, очевидно, несложной задачей, а вот все компьютерные системы, моделирующие этот процесс, до сих пор имеют возможности, существенно уступающие возможностям самого обычного первоклассника. Созданы системы (т.е. машинные модели) понимания речи, доказавшие свою исключительную практическую пользу. Например, можно построить компьютеризованную систему заказа авиабилетов, ведущую диалог с пользователем на обычном разговорном языке, и такая система будет правильно отвечать на большинство вопросов пользователя. Необходимость в наличии систем понимания речи становится очевидной, если вспомнить о возможностях, возникающих в связи с ускорением развития средств телекоммуникации. Все более привычными становятся электронные банковские расчеты в интерактивном режиме, и постоянно растет интерес к разработке автоматизированных агентов, способных осуществлять поиск и анализировать огромные объемы данных в Internet. Однако все такие приложения имеют довольно узкие области применения. Ограничение области применения позволяет снизить уровень сложности разработки за счет сокращения размера словаря и упрощения задачи понимания смысла запроса. И хотя число узкоспециальных приложений для распознавания разговорного языка за последние несколько лет заметно выросло, мы все еще не можем сказать, когда же машинные возможности понимания речи приблизятся к тому уровню понимания, на котором находится человек. Разговорный язык является слишком выразительной формой коммуникации, ставящей перед ученым и программистом массу проблем. Несколько примеров представлено в табл. 7.7.

Давайте немного поговорим о том, как выяснить значение предложения или разрешить неоднозначность утверждения, чтобы подчеркнуть, что система понимания речи сильно зависит от наличия в ее распоряжении большой базы данных реальных знаний. Многовариантность возникает там, где мы не можем быть уверены, что наша интерпретация ситуации оказывается правильной, но общие знания дают нам определенную степень уверенности, а также дают возможность менять интерпретацию в свете информации, поступающей в дальнейшем.

Таблица 7.7. Некоторые трудности обработки разговорного языка

Тип проблемы	Пример	Объяснение
Передача одинакового смысла разными по форме высказываниями	1. Стюарт разбил вазу 2. Ваза была разбита Стюартом	(1) выражено в активной форме (2) является пассивной формой (1)
Роль предлога "с"	1. Незнакомка толкнула мальчика с девочкой 2. Незнакомка толкнула мальчика с собакой	В (1) предлог "с" означает объединение пострадавших, тогда как в (2) предлог "с" добавляет описание мальчика
Многие слова имеют несколько значений	1. Уход за ребенком 2. Уход со сцены	В (1) "уход" обозначает постоянный присмотр и заботу, тогда как в (2) "уход" обозначает однократное конкретное действие
Смысловое согласование слов	1. Угон автомобиля является опасностью 2. Угон автомобиля является опасным	В (1) имеется в виду опасность для владельца, тогда как в (2) имеется в виду опасность для преступника
Порядок слов	1. Музей сохранил доход 2. Доход сохранил музей	В (1) может иметься в виду, что был сохранен доход, а в (2) — что был сохранен музей

Как и в случае большинства других сложных задач, задача реализации системы понимания речи на компьютере разбивается на несколько уровней. Тремя уровнями, рассматриваемыми в дискуссиях, посвященных системам понимания речи, являются *синтаксический* уровень, *семантический* уровень и *прагматический* уровень. Синтаксический уровень касается того, как слова объединяются в структуры, называемые выражениями, и как выражения объединяются в структуры, называемые предложениями. Для проверки того, является ли некоторое предложение допустимым (т.е. грамматически правильным) и для разбиения предложения на составные части, обычно используются правила грамматики. Семантический и прагматический уровни касаются значения предложений. На семантическом уровне выделяется содержание (суть предложения), и для выражения содержания часто используются определенные логические формы. Например, предложения

Джон ударил по мячу.

Ударил ли Джон по мячу?

имеют одно и то же содержание, которое в языке Prolog может быть представлено в виде:

ударил(джон, мяч).

Имея возможность автоматизировать синтаксический и семантический анализ, мы можем утверждать:

Джон ударил по мячу. ударил(джон, мяч)

Затем мы можем спросить:

Кто ударил по мячу? ударил(Х, мяч)

По чему ударил Джон? ударил(джон, Х)

Кто по чему ударил? ударил(Х, Y)

Прагматический анализ касается интерпретации предложения в контексте. Так, вопрос об интерпретации местоимения “Он” в предложении

Он забил гол.

может быть решен только в контексте (т.е. с использованием соседних предложений). Чтобы ответить на вопрос “Кто забил гол?”, мы должны решить, на кого именно указывает местоимение “Он”. Намерения говорящего тоже контекстно-зависимы. Предложение

Можете ли вы приготовить чай?

часто означает просьбу приготовить чай, но может иметь и буквальный смысл: “Умеете ли вы готовить чай?”. Контекст также используется для того, чтобы ограничить смысл определенных слов. Слово “любить” в обыденном использовании имеет смысл, выражаемый в форме

любит(человек, объект).

Это смысл допускает выражения типа “Джон любит Мери”, “Джон любит работу” и “Джон любит шоколад”. В контексте отношений между людьми “любит” может быть ограничено формой

любит(человек, человек),

и в этом смысле мы можем предполагать, что “Венера” в

любит(джон, венера)

является именем женщины, а не названием планеты. В данном случае ограничение решает потенциальную неоднозначность между тем, что может обозначать “Венера” — имя человека или название планеты.

Давайте теперь вкратце рассмотрим сам процесс синтаксического и семантического анализа.

7.5.1. Синтаксический анализ

Фрагмент текста складывается из предложений, а каждое предложение компонуется из выражений, которые могут содержать подвыражения и, в конечном счете, состоят из слов. Грамматика определяет способ, в соответствии с которым слова и выражения могут объединяться в предложения. Грамматика может быть представлена с помощью любого из множества языков представлений, но чаще всего для этого используют формат правил вывода. Пример грамматики, представленной с использованием правил вывода, показан на рис. 7.5. На рис. 7.6 показана структура предложения “the dog ran across the road” английского языка (в переводе оно означает “собака бежала через дорогу”).

S → NP VP NP V
NP → D AP D N NP PP
PP → P NP
VP → V NP V PP
AP → A AP A N

Рис. 7.5. Пример грамматики, выраженной правилами вывода. S означает предложение, N – имя существительное, A – прилагательное, P – предлог, D – определитель (артикл), V – глагол, NP – именная группа, VP – глагольная группа, AP – группа прилагательного, PP – предложная группа

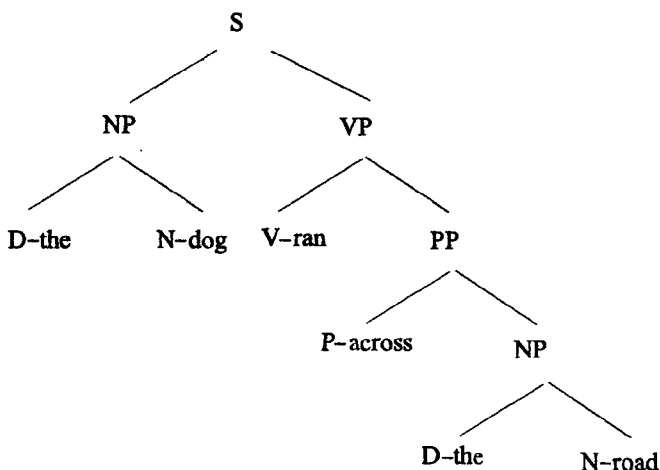


Рис. 7.6. Дерево синтаксического анализа для предложения “The dog ran across the road”

Предложение анализируется с целью выделения входящих в его структуру выражений и с целью его проверки на соответствие грамматике. Для анализа предложения требуется также словарь. Словарь определяет категории слов (N, V, P, A, D). Такой анализ может быть интерпретирован как поиск, при котором поддерживается стек дублирования для резервного копирования состояний, чтобы имелась возможность вернуться из тупикового состояния.

Пример 7.3.

Иллюстрируйте то, как можно в виде поиска провести синтаксический анализ предложения “The dog ran across the road”. Используйте следующую грамматику:

$S \rightarrow NP VP$
 $NP \rightarrow D AP | D N$
 $PP \rightarrow P NP$
 $VP \rightarrow V PP$
 $AP \rightarrow A N$

и следующий словарь:

$D \rightarrow \text{the}$ $N \rightarrow \text{dog} | \text{road}$ $V \rightarrow \text{ran}$ $P \rightarrow \text{across}$

Решение 7.3.

Процесс показан в табл. 7.8, а генерируемые при этом состояния — на рис. 7.7.

Таблица 7.8. Синтаксический анализ предложения “The dog ran across the road”, рассматриваемый как поиск

Текущая попытка	Стек слов	Стек посещений	Стек дублирования
$S \rightarrow NP VP$	the dog ran across the road	NP VP	
$NP \rightarrow D AP$	the dog ran across the road	D AP VP	$NP \rightarrow DN$
$D \rightarrow \text{the}$	dog ran across the road	AP VP	$NP \rightarrow DN$
$AP \rightarrow A N$	dog ran across the road	VP	$NP \rightarrow DN$
dog не относится к категории A, поэтому возврат			
$NP \rightarrow D N$	the dog ran across the road	D N VP	
$D \rightarrow \text{the}$	dog ran across the road	N VP	
$N \rightarrow \text{dog}$	ran across the road	VP	

Окончание табл. 7.8

Текущая попытка	Стек слов	Стек посещений	Стек дублирования
VP → V PP	ran across the road	V PP	
V → run	across the road	PP	
PP → P NP	across the road	P NP	
P → across	the road	NP	
NP → D AP	the road	D AP	NP → DN
D → the	road	AP	NP → DN
AP → A N	road	A N	NP → DN
road не относится к категории A, поэтому возврат			
NP → D N	the road	D N	
D → the	road	N	
N → road			

Алгоритмы синтаксического анализа изучались достаточно интенсивно, поскольку синтаксический анализ является основой многих вычислительных систем, будь то компиляция программ или проверка синтаксиса запроса к базе данных. Предложенный здесь пример предложения выглядит тривиальным, но любая реальная система понимания речи использует более сложную грамматику, позволяющую понимать, например, согласование множественного числа и времен. Обсуждения этих усовершенствований остаются за рамками нашей книги, но заинтересовавшиеся читатели найдут соответствующие указания в конце главы в разделе дополнительной литературы.

7.5.2. Семантический анализ

Семантический анализ часто реализуется как этап синтаксического анализа в результате добавления к правилам грамматики соответствующих свойств и использования сравнения образцов. Здесь мы имеем возможность сказать только несколько слов о том, как выделяется суть предложения в форме языка Prolog. Грамматика

```

S → NP VP
NP → N
N → name
VP → V NP
name → мери | джон
verb → любит

```

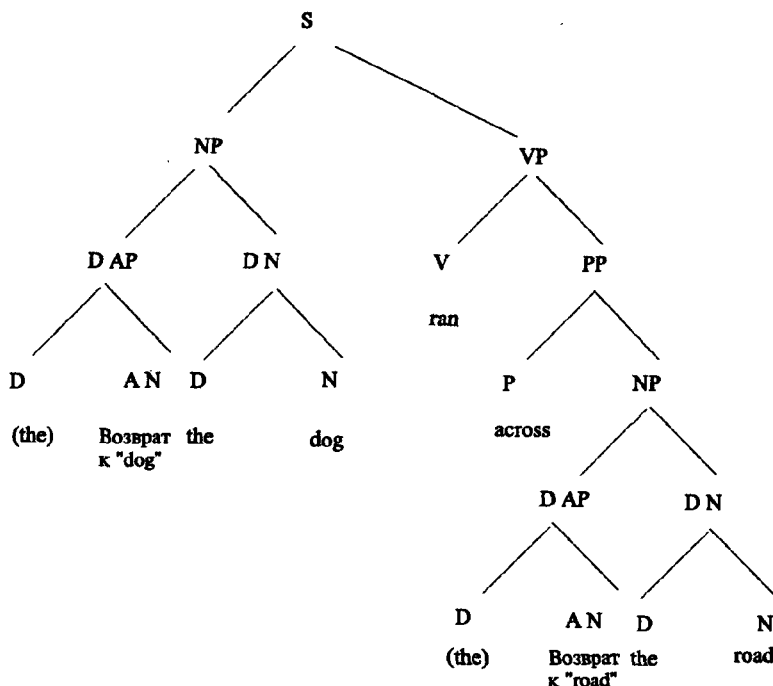


Рис. 7.7. Состояния, генерируемые при синтаксическом анализе предложения "The dog ran across the road"

допускает предложение "Джон любит Мери". В языке Prolog оно имеет следующую форму.

любит(джон, мери)

Чтобы выделить эту форму в процессе синтаксического анализа, грамматика дополняется следующими свойствами.

$S(\text{sem-vp sem-np}) \rightarrow NP(\text{sem-np}) VP(\text{sem-vp})$

$NP(\text{sem-np}) \rightarrow N(\text{sem-np})$

$N(\text{sem-n}) \rightarrow (\text{sem-n})$

$VP(\text{sem-v sem-np}) \rightarrow V(\text{sem-v}) NP(\text{sem-np})$

$\text{name}(\text{sem-n}) \rightarrow \text{мери}(\text{"мери"})$

$\text{name}(\text{sem-n}) \rightarrow \text{джон}(\text{"john"})$

$\text{verb}(\text{sem-v}) \rightarrow \text{любит}(\lambda y. \lambda x. \text{любит}(x, y))$

Указанный здесь глагольный элемент (verb) содержит выражение, называемое λ -исчислением. Оно используется для того, чтобы выполнить простую операцию, называемую λ -редукцией, и упростить выражение. Например, выражение в форме

$$((\lambda \times Px)a)$$

может быть редуцировано (т.е. сведено) к “Pa”. Поэтому выражение $\lambda y.\lambda x.\text{любит}(x, y)$, сформулированное в виде $(\lambda y.\lambda x.\text{любит}(x, y))$ “мери”, сводится к $\lambda x.\text{любит}(x, \text{мери})$, а $((\lambda x.\text{любит}(x, \text{мери}))$ “джон”) сводится к $\text{любит}(\text{джон}, \text{мери})$.

Такая λ -редукция выполняется в порядке слева направо, и ее целью является приведение выражения к более легко читаемой (или распознаваемой) форме.

В случае нашего простого примера можно представить сравнения свойств как серию вызовов функций. Левая сторона правила рассматривается как функция, а правая сторона выполняет серию вызовов других функций. Эти операции продолжаются рекурсивно до тех пор, пока слову не будет найдено соответствие и не будет возвращена строка.

Предложение “Джон любит Мери” будет проанализировано с использованием “псевдо-С” нотации и с применением λ -редукции там, где это необходимо. Учтите то, что некоторые скобки в грамматических выражениях были опущены с целью сохранения простоты записи.

вызов S с аргументом “джон любит мери”

возвращает $(\lambda x.\text{любит}(x, \text{мери}) \text{ джон}) = \text{любит}(\text{джон}, \text{мери})$

S(sem-vp sem-np) {

NP(sem-np)

в результате вызова NP в конечном итоге возвращается джон

VP(sem-vp)

в результате вызова VP возвращается

$(\lambda y.\lambda x.\text{любит}(x, y) \text{ мери}) = (\lambda x.\text{любит}(x, \text{мери}))$

}

NP(sem-np) {

N(sem-np)

}

N(sem-n) {

name(sem-n)

}

name(sem-n) {

if “джон”

return sem-n = “джон”

if “мери”

return sem-n = “мери”

}

VP(sem-v sem-np) {

V(sem-v)	в результате вызова V возвращается $(\lambda y.\lambda x.\text{любит}(x, y))$
NP(sem-np)	возвращается меры

```

}
V(sem-v) (
  sem-v =  $(\lambda y.\lambda x.\text{любит}(x, y))$ 
)

```

7.6. Символьные связи нейронных сетей

Идея использования нейронных сетей для выполнения задач, традиционно относящихся к области символьных систем искусственного интеллекта, апеллирует к интуиции инженеров и не только потому, что абстрактные архитектуры искусственного мозга могут выдавать нестандартные решения, но также и потому, что нейронные сети имеют целый ряд других весьма привлекательных свойств. Нейронные сети обучаются выполнению задачи с помощью адаптации к предлагаемым сети стимулам. Система, основанная на обучении, имеет возможность выводить знания автоматически, а также обнаруживать знания, которые являются специфическими для конкретной задачи и которые трудно представить в виде набора правил (например, как ехать на велосипеде). Сеть редко предлагает однозначный ответ, она обычно обеспечивает ответ в виде оценки, но для решения практических задач оценочные ответы кажутся как раз более подходящими. Нейронные сети могут также демонстрировать плавное снижение своей эффективности, сохраняя способность выполнять задачи (хотя и на худшем уровне), если часть структуры сети оказывается поврежденной. Нейронные сети могут иногда сохранять свою работоспособность, даже если какая-то часть архитектуры сети перестает функционировать вообще. Наконец, нейронные сети могут быть реализованы в виде массовых параллельных структур, которые потенциально могут предложить более эффективную производительность по сравнению с машинами, использующими последовательную архитектуру.

Символьные системы имеют свои преимущества. Средством коммуникации являются символы, и в процессе коммуникации происходит передача большого объема знаний. Например, если кому-то скажут, что он погибнет, если коснется рельсов, по которым идут поезда метро, то, за редким исключением, человек серьезно отнесется к такому предупреждению. Знание, выраженное в виде правил, может сохранить жизнь в ситуациях, которые ранее не были испытаны на практике. Пилот самолета, непреднамеренно вводящий самолет в штопор, лицом к лицу сталкивается с ситуацией, несущей угрозу жизни, и хотя практика выхода из штопора может помочь справиться с ситуацией опытному пилоту, неопыт-

ному пилоту может помочь знание соответствующих правил. Знание, сообщаемое в символьной форме, может также ускорить обучение. Трудным этапом подготовки любого пилота является обучение посадке. Хороший инструктор может предложить эмпирические правила, способные помочь пилоту-новичку и может повторить или подкорректировать инструкции, обнаружив неправильное или плохое их выполнение.

Итак, и символьная парадигма, и парадигма нейронных сетей имеют свои преимущества с точки зрения построения интеллектуальных систем. Поэтому неудивительным является тот факт, что проводилось (и проводится сегодня) немало исследований, посвященных возможности налаживания связей между традиционным или символьным искусственным интеллектом и нейронными сетями. Эти связи очень разнообразны, но главным вопросом для многих приверженцев нейронных сетей является, по-видимому, вопрос о том, могут ли быть созданы нейронные сети, способные решать когнитивные задачи самого высокого уровня, какими являются, например, понимание речи и планирование.

Материал, представленный в следующей главе, инспирирован философскими дебатами, которые велись на протяжении второй половины 1980-х годов (см., например, [Fodor, Pylyshyn, 1988]). Первые приложения нейронных сетей, последовавшие за возрождением интереса к нейронным сетям в 1980-х годах, касались проблем распознавания, относящихся к низкому уровню, и выполняли, по существу, построение отображения множества начальных данных во множество подходящих ответов. А поскольку велись оживленные дискуссии в отношении нейронных сетей как когнитивных моделей, неудивительно и то, что некоторые философы и когнитивные психологи начали интересоваться возможностями сетей первой генерации. Было выяснено, что часть критических замечаний была основана на незнании того, как добиться от нейронной сети необходимого качества выполнения задания. Следует заметить, что критические вопросы все еще ждут своих достаточно ясных ответов, но мы надеемся, что с этим не будет проблем по мере расширения наших знаний в области нейронных сетей.

Тремя характерными особенностями, отвечающими за успех символьной парадигмы, являются композиционные структуры, чувствительная обработка структур и обобщение. Все эти три свойства оказываются тесно связанными.

Композиционные структуры могут быть определены как иерархические структуры, в которых вся структура складывается из отдельных частей. Типичным примером таких структур являются деревья, а также наследственные и агрегированные модели, используемые инженерами. Композиция структур является основой любой формы разработки конструкций: при попытках решения любой сложной задачи такая задача обычно разбивается на части. Мы уже видели в разделе 7.5, что компози-

ционный анализ играет большую роль в системах понимания речи. Структура выражений предложения представлялась в виде иерархии его частей, и семантический анализ предполагал, что смысл всего предложения может быть найден как комбинация содержания частей.

Трансформация $\neg(P \wedge Q)$ в его логический эквивалент $(\neg P \vee \neg Q)$ является примером операции, чувствительно зависящей от структуры. Если P и Q заменить на R и S , то операция остается правомерной, так как структурной формой является суть: $\neg(R \wedge S)$ трансформируется в логический эквивалент $(\neg R \vee \neg S)$. Данная операция может быть также представлена в виде более сложных выражений, поскольку все выражение может быть разбито на части. Например, $\neg((P \vee R) \wedge Q)$ все равно имеет форму $\neg(A \wedge Q)$, поэтому его логическим эквивалентом является $(\neg(P \vee R) \vee \neg Q)$.

Абстракция обобщенных выражений тоже является естественной частью символической парадигмы. Аргументы в выражении любит(одушевленный, объект) допускают любую комбинацию объектов, подходящих для рассматриваемого отношения. Это позволяет системе демонстрировать регулярное поведение. Если система регулярна, она должна допускать семантическое (и синтаксическое) обобщение. Регулярная система, понимающая “джон любит мери”, поймет также и “мери любит джона”.

Наше обсуждение нейронных сетей в следующей главе сосредоточится именно на этих основных свойствах символических систем.

7.7. Резюме

В области традиционного искусственного интеллекта значительные усилия тратятся на исследование представлений. Ключевым компонентом любой интеллектуальной системы является способность адаптироваться к изменениям среды, т.е. способность обучаться. Традиционные системы искусственного интеллекта для представления знаний используют символические структуры, так что вычисления могут рассматриваться как последовательность операций, применяемых к этим структурам в процессе поиска решения. Ниже перечислены некоторые ключевые моменты, на которые опирается традиционный искусственный интеллект.

- Знание представляется в явном виде.
- Структурные компоненты могут объединяться для того, чтобы строить еще более сложные структуры. Это значит, что более крупные структуры могут быть скомпонованы из малых структур, и, наоборот, эти более крупные структуры могут быть разделены на меньшие.
- Символические представления позволяют обобщение и абстрактное описание знаний.

7.8. Дополнительная литература

Если для вас тема искусственного интеллекта является новой, то хорошей книгой для знакомства с предметом является книга [Cawsey, 1998], вышедшая в той же серии книг, что и та, которую вы держите в своих руках. Философское введение в область искусственного интеллекта [Copeland, 1993] предлагает общий обзор хорошего уровня. Текст книги [Dean *et al.*, 1995] является хорошим изложением предмета с рядом интересных примеров, но это изложение слишком тяготеет к языку Lisp. Превосходным и всеобъемлющим изложением вопросов, относящихся к области искусственного интеллекта, является книга [Russell, Norvig, 1995].

Одним из наиболее амбициозных и, вероятно, крупнейшим из современных проектов, посвященных искусственному интеллекту, является проект CYC. Проект CYC представляет собой массивную базу знаний, предназначенную для хранения знаний, которые могут потребоваться для того, чтобы получить доступ к информации, содержащейся в энциклопедиях, и придать ей смысл. CYC хранит большие объемы тех общих знаний, которые мы используем каждый день и которые, казалось бы, не требуют никаких усилий для их применения. Например, когда вам говорят о чьем-то дне рождения, вы знаете, что день рождения этого человека отмечается в тот же день каждый год. Но CYC содержит не только это. Значительный объем знаний CYC был запрограммирован командой инженеров-специалистов, но теперь CYC находится на такой стадии, что уже может обучаться огромным объемам знаний самостоятельно с помощью чтения текста. CYC требуется определенная помощь в понимании текста, но главная цель команды CYC — достижение понимания естественного языка. Лучшим источником информации о CYC является Web-страница CYC:

<http://www.cyc.com/>

7.9. Упражнения

1. Покажите, что выражение

$$((P \Rightarrow Q) \wedge R) \Rightarrow S$$

будет проанализировано как допустимое в соответствии с грамматикой, представленной в табл. 7.2.

2. Используя таблицу истинности, покажите, что $\neg((P \vee R) \wedge Q)$ эквивалентно $(\neg(P \vee R) \vee \neg Q)$.
3. На рис. 7.2 показан пример использования продукционной системы для доказательства цели при следующих правилах и при условии, что T, W и P есть ИСТИНА.

- 1 $P \wedge Q \wedge R \Rightarrow \text{цель}$
- 2 $S \Rightarrow R$
- 3 $T \wedge W \Rightarrow Q$
- 4 $T \Rightarrow S$

Используйте правила вывода логики высказываний, чтобы доказать цель.

4.

- $$\begin{aligned} S &\rightarrow NP VP \mid NP V \\ NP &\rightarrow D AP \mid D N \mid NP PP \\ PP &\rightarrow P NP \\ VP &\rightarrow V NP \mid V PP \\ AP &\rightarrow A AP \mid A N \end{aligned}$$

Используя эту грамматику, постройте дерево синтаксического анализа для:

- (а) “The little dog barked” (маленькая собака залаяла);
- (б) “The horse jumped over the fence” (лошадь перепрыгнула через изгородь).
5. Выпишите состояния поиска, генерируемые в ходе синтаксического анализа предложений из упражнения 4.
6. Как нужно изменить грамматику из упражнения 4, чтобы предложение “John found the book” (Джон нашел книгу) оказалось допустимым?
7. Имеется база данных Prolog:

- $$\begin{aligned} &\text{часть(закрылок, крыло)} \\ &\text{часть(элэрон, крыло)} \\ &\text{часть(крыло, самолет)} \end{aligned}$$

Перефразируйте следующие вопросы и покажите, как Prolog ответит на них.

- (а) часть(закрылок, крыло).
- (б) часть(крыло, X).
- (в) часть(X, крыло).

8. Предположим, что к коду Prolog из упражнения 7 добавляется

часть(X, Y) :- часть(Z, Y), часть(X, Z)

- (а) Как программа ответит на часть(закрылок, самолет)? Трассируйте соответствующий процесс сравнения.
- (б) Не видите ли вы каких-либо потенциальных проблем, которые могут возникнуть с указанным выше правилом?

Глава 8

Синтез символов с помощью нейронных сетей

Задача. Выяснение некоторых связей между традиционным искусственным интеллектом и нейронными сетями.

Цели. Вы должны понять:

в чем состоят потенциальные преимущества комбинированного подхода на основе символьных представлений и нейронных сетей;
что такое локальные и распределенные представления;
какие подходы используются исследователями нейронных сетей для решения проблем понимания речи;
что такое композиционное и систематическое представления;
что такое “обоснование символов”.

Требования. Знакомство с материалом глав 1–7.

8.1. Нейронные сети в символьной форме

В этой главе мы не рассматриваем новый искусственный интеллект *в целом*, а только аспект, связанный с нейронными сетями, а также некоторые показательные примеры новейших исследований, иллюстрирующие фундаментальные понятия этой области. Вместо термина “нейронная сеть” часто используется термин “*коннекция*”, а термин “*коннекционизм*” используется для обозначения универсальной парадигмы нейронных сетей. Приверженцы коннекционизма — ученые, стремящиеся “обуздать” вычислительный потенциал нейронных сетей, не касаясь непосредственно биологического реализма, в отличие от другого направления исследований нейронных сетей, где нейронные сети используются в попытке найти объяснение процессов, происходящих внутри мозга человека.

Существует несколько заслуживающих внимания моделей коннекций (см. список дополнительной литературы в разделе 8.8), разработанных с целью кодирования и последующей обработки символьных структур. В этой главе мы сначала рассмотрим архитектуру нейронной сети, называемую *рекурсивной автоассоциативной памятью* (RAAM — Recursive Autoassociative Memory). Для подробного обсуждения сети RAAM есть несколько причин. Во-первых, сеть RAAM может быть описана почти или даже вообще без использования математики. Во-вторых, появление сети RAAM — серьезное предупреждение для критики, заявляющей, что нейронные сети не могут вести себя композиционным и систематическим образом. Кроме того, сеть RAAM вызвала огромный интерес со стороны сообщества исследователей нейронных сетей, уже проведено множество исследований и имеется обширная литература на эту тему. Наконец, что, возможно, более важно, архитектуру этой сети можно использовать при исследовании некоторых ключевых вопросов.

После разработки архитектуры сети RAAM и выяснения некоторых вопросов, касающихся представлений, были предложены две модели понимания речи. Эти модели имеют достаточно сложную архитектуру, и мы не ставим своей задачей обеспечить читателю глубокое понимание реализаций этих моделей. Целью представления здесь этих и следующих за ними моделей компьютерного общения является информирование читателя о том, что специалисты по нейронным сетям уже пытаются решать некоторые весьма сложные проблемы, относящиеся к сфере традиционного искусственного интеллекта. И хотя сегодня такие исследования находятся на ранней стадии, эти модели дают возможность почувствовать, как в будущем на основе базовых нейронных структур могут разрабатываться сложные системы. Мы на самом деле находимся в самом начале разработки “искусственных нейронных систем”, и первые результаты говорят о том, что нас при этом еще могут ожидать разные сюрпризы.

В этой же главе будут рассмотрены важные, но трудные для понимания вопросы *абстрактного обобщения и обоснования символов*.

Читатель не должен сдаваться, если материал этой главы покажется ему слишком трудным. Некоторые из обсуждаемых ниже принципов сложны для восприятия, а реальное их понимание может быть получено только в результате изучения рекомендованной литературы. Эта глава и предполагалась как более трудная по сравнению с остальными, но ее главная задача — ознакомить читателя с вопросами, которые, несомненно, являются важными и которые постепенно привлекают к себе то внимание, которого они заслуживают.

8.2. Рекурсивная автоассоциативная память

Архитектура сети RAAM была предложена Поллаком [Pollack, 1990]. Целью RAAM является представление символьных структур средствами нейронных сетей. Символьные структуры представляют собой деревья фиксированной валентности. Валентностью называют число ветвей, выходящих из каждой вершины, и фиксированная валентность означает, что все вершины должны иметь одно и то же число выходящих из них ветвей.

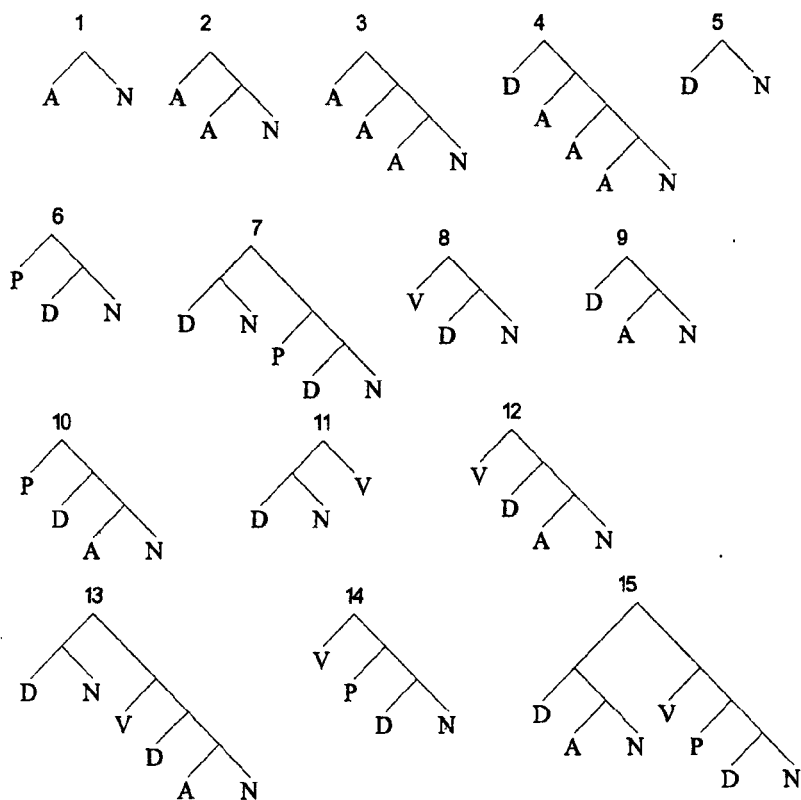
На рис. 8.1 показаны 15 бинарных деревьев (валентность 2). Эти деревья идентифицируют структуры выражений, генерируемых в соответствии со следующими правилами грамматики:

$$\begin{aligned} S &\rightarrow NP VP | NP V \\ NP &\rightarrow D AP | D N | NP PP \\ PP &\rightarrow P NP \\ VP &\rightarrow V NP | V PP \\ AP &\rightarrow A AP | A N \end{aligned}$$

где используемые символы имеют те же значения, что и на рис. 7.5.

Сеть RAAM является по существу автоассоциативной сетью с обратным распространением ошибок (см. главы 2 и 4). Во входном и выходном слоях RAAM элементы организованы в поля, где каждое поле содержит одинаковое число элементов. Число полей определяется валентностью кодируемых деревьев, а число элементов в скрытом слое соответствует числу элементов одного поля. На рис. 8.2 показано два примера соответствующей архитектуры.

Давайте проигнорируем способ обучения сети RAAM и рассмотрим только то, как эта сеть создает представление и снова разбивает его на составные части. Если взять сеть RAAM, обученную представлять некоторое множество деревьев, то эту сеть можно рассматривать как два автомата: первый слой весовых значений является автоматом для создания представлений деревьев, а второй слой весовых значений — автоматом, берущим созданное представление и разбивающим это представление на составные части. Первый автомат будем называть *кодером*, а второй — *декодером*. Представление некоторой структуры в сети RAAM порождается значениями активности скрытого слоя. В процессе создания представления всего дерева RAAM генерирует представления для каждой внутренней вершины (т.е. для каждого поддерева). Так, кодирование $(DN)(P(DN))$ ведет к появлению представлений для (DN) и $(P(DN))$, генерируемых в процессе выполнения. Деревья являются рекурсивными структурами, и RAAM формирует их представления некоторым рекурсивным образом с помощью обратной подпитки входного слоя ранее сформированными представлениями частей, происходящей в определенные моменты времени. Кодирование структуры $(DN)(P(DN))$ показано на рис. 8.3, а ее декодирование — на рис. 8.4.



- | | |
|-----------------------------|---------------------|
| 1. (AN) | 2. (A (AN)) |
| 3. (A (A (AN))) | 4. (D (A (A (AN)))) |
| 5. (P (DN)) | 6. ((DN)(P(DN))) |
| 7. (V (DN)) | 8. (D (AN)) |
| 9. (P (D (AN))) | 10. (D (AN)) |
| 11. ((DN) V) | 12. (V (D (AN))) |
| 13. ((DN) (V (D (AN)))) | 14. (V (P (DN))) |
| 15. ((D (AN)) (V (P (DN)))) | |

Рис. 8.1. Эти деревья взяты из [Pollack, 1990]. Деревья показывают структуру выражений, из которых состоят предложения (или части предложений). Обратите внимание на то, что многие деревья являются поддеревьями (т.е. частями) других деревьев. Например, дерево 1 является частью дерева 2, а дерево 2 является частью дерева 3. Точно так же, деревья 8 и 14 являются частями дерева 15

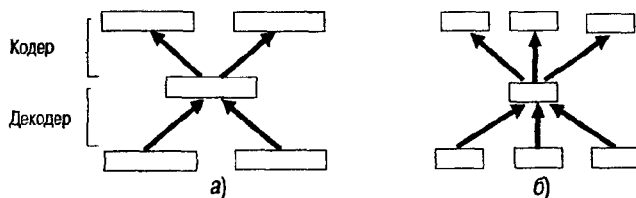
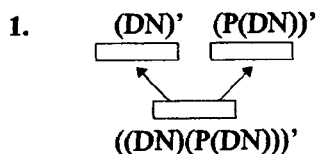


Рис. 8.2. Архитектура а представляет двоичную сеть RAAM, а б – троичную

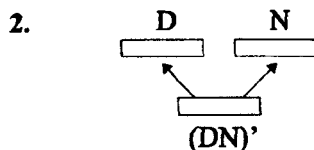


Рис. 8.3. Кодирование $(DN)(P(DN))$. Одна и та же сеть показана на трех различных стадиях кодирования представления

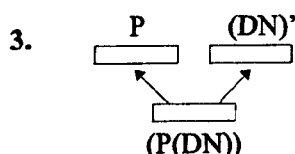
Терминальные символы (D , A , N , V и P) представляются в сети векторами. Как правило, используются ортогональные векторы, и в данном случае это означает, что каждый символ представляется вектором с пятью элементами (поскольку имеется пять символов), и у каждого из соответствующих векторов один элемент равен 1, а все остальные равны 0 (все векторы должны быть разными и только с одним битом, равным 1). Примером подходящего набора векторов будут следующие векторы:



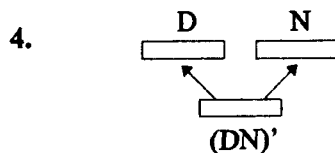
Представление для всего дерева
подается на вход декодера



$(DN)'$ со стадии 1 рассматривается
и декодируется снова в D и N



$(P(DN))'$ со стадии 1 рассматривается
и декодируется снова в P и $(DN)'$



$(DN)'$ со стадии 1 рассматривается
и декодируется снова в D и N

Рис. 8.4. Декодирование $(DN)(P(DN))'$. Одна и та же сеть показана на четырех различных стадиях декодирования представления

D	1 0 0 0 0,
N	0 0 1 0 0,
P	0 0 0 0 1,
A	0 1 0 0 0,
V	0 0 0 1 0.

Размерность (число битов) векторов, представляющих символы, определяет минимальное число элементов, составляющих поле RAAM. Скрытый слой, в конечном итоге, порождает сжатое представление, поскольку число элементов в этом слое меньше числа входных элементов. Но норма сжатия, которая при этом может быть достигнута, зависит от конкретного набора учебных данных, в связи с чем может возникнуть необходимость дополнить соответствующие терминальным символам векторы нулями. Например, сеть RAAM, используемая для представления деревьев, изображенных на рис. 8.1, имела архитектуру типа 20-10-20. Таким образом, каждое поле содержало десять элементов, а это значит, что векторы терминальных символов требовалось дополнить пятью нулями. Подобно большинству других сетей, число скрытых элементов определяется из эксперимента (в ходе которого находится архитектура, обеспечивающая удовлетворительное решение задачи).

При декодировании требуется проверка, позволяющая выяснить, чем является появляющийся в поле выходного слоя образец — терминальным символом или образцом, требующим сохранения для дальнейшей обработки в скрытом слое по обратной связи. Для такой проверки есть несколько возможностей, но в случае ортогональных символов проще всего проверить, будет ли один элемент равен единице, а все остальные — нулю. Если наличие терминального символа не подтверждается, то вывод может быть снова пропущен через скрытый слой.

8.2.1. Обучение RAAM

Процесс обучения в данном случае соответствует процессу обучения любой автоассоциативной сети, но с требованием возврата во входной слой ранее полученной информации. Правила такой обратной связи такие же, как для и конструкции, обсуждавшейся выше. Терминальным символам соответствуют векторы, во входном множестве фиксированные, но так как весовые коэффициенты в ходе обучения постоянно корректируются, представления всех других векторов должны меняться. Например, скрытое представление для (D N) в ходе обучения изменяется, так как постоянно меняются весовые значения. Такое изменение во множестве учебных образцов называется эффектом *движущейся цели*. Когда сеть приближается к точке сходимости, изменения движущихся целей становятся очень маленькими.

Что касается реализации сети RAAM, единственной дополнительной сложностью по сравнению с обычными автоассоциативными сетями с обратным распространением ошибок является необходимость возврата ранее сформированных в скрытом слое образцов снова во входной слой. Этот вопрос легко решается, если использовать структуру стека данных для хранения информации о внутренних узлах. Представление последовательностей с помощью RAAM оказывается даже более простым. Например, последовательность, соответствующая слову BRAIN, может быть представлена в виде дерева, показанного на рис. 8.5. Терминальный символ NIL используется просто как пробел. Кодирование начинается, как обычно, снизу вверх, и В подается на рассмотрение левому входному полю, а NIL — правому входному полю. Ввиду того, что сжатые представления подаются по обратной связи только правому входному полю, левое и правое входные поля (и, таким образом, соответствующие выходные поля) могут содержать разное число элементов: ограничение заключается только в том, что скрытый слой и правые входное и выходное поля должны быть одного и того же размера. Для последовательности нет необходимости хранить представления в стеке, так как ранее сжатые элементы подаются обратно во входной слой.

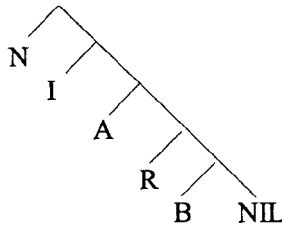


Рис. 8.5 Представление слова BRAIN в виде дерева

Пример 8.1.

1. Продемонстрируйте порядок кодирования и декодирования бинарного дерева

$((D (A N)) (V (P (D N))))$.

2. Для предложенного ниже учебного множества выясните, сколько деревьев научится представлять сеть RAAM, если использовать это множество?

$(D (A (A (A N))))$
 $((D N) (P (D N))$
 $(V (D N))$
 $(P (D (A N)))$
 $((D N) V)$
 $((D N) (V (D (A N))))$
 $((D (A N)) (V (P (D N))))$

Решение 8.1.

1. Порядок кодирования показан в табл. 8.1, а порядок декодирования — в табл. 8.2.
2. Весь соответствующий набор деревьев показан на рис. 8.1.

Таблица 8.1. Порядок кодирования бинарного дерева из примера 8.1

Левая ветвь	Правая ветвь	Скрытое представление
D	N	$(D N)'$
P	$(D N)'$	$(P (D N))'$
V	$(P (D N))'$	$(V (P (D N)))'$
A	N	$(A N)'$
D	$(A N)'$	$(D (A N))'$
$(D (A N))'$	$(V (P (D N)))'$	$((D (A N)) (V (P (D N))))'$

Таблица 8.2. Порядок декодирования бинарного дерева из примера 8.1

Скрытое представление	Левая ветвь	Правая ветвь
$((D(A N))(V(P(D N))))'$	$(D(A N))'$	$(V(P(D N)))'$
$(D(A N))'$	D	$(A N)'$
$(A N)'$	A	N
$(V(P(D N)))'$	V	$(P(D N))'$
$(P(D N))'$	P	$(D N)'$
$(D N)'$	D	N

8.3. Представления нейронных сетей

Архитектура RAAM использует структуры переменного размера (т.е. деревья разной длины) и отображает их в векторы фиксированной длины. Такое отображение имеет ряд определенных свойств, которые мы сейчас и рассмотрим.

8.3.1. Локальные и распределенные представления

Локальное представление возникает тогда, когда сеть устроена таким образом, что отдельные элементы означают конкретные понятия. Например, один элемент может обозначать “собака”, другой — “лошадь”, третий — “кролик”, четвертый — “волк” и т.д. Но чаще представления для одного понятия бывают распределенными. Идея распределенного представления иллюстрируется схемой на рис. 8.6. Распределенное представление до известной степени оказывается толерантным к дефектам, так как повреждение одного или нескольких элементов не обязательно означает, что сеть утратит все представление соответствующего понятия. В распределенном представлении элемент только участвует в представлении понятия (и таких элементов обычно несколько). Когда понятие представлено в распределенном виде, оно выглядит как распределенный набор значений активности. Левое поле входных элементов в последовательной сети RAAM действует как локальное представление в случае ортогональных терминальных векторов: только один элемент в терминальном векторе имеет значение 1, поэтому только один элемент перейдет во “включенное” состояние. Сжатые представления, создаваемые скрытыми элементами, обычно оказываются распределенными в том смысле, что, например, одна буква соответствующей слову последовательности представ-

ляется несколькими скрытыми элементами. Для представления понятий распределенная архитектура оказывается более экономичной, чем локальная. Например, сеть, имеющая n двоичных элементов, при локальном подходе может представить только n понятий, а при использовании распределенной архитектуры можно представить порядка 2^n понятий.

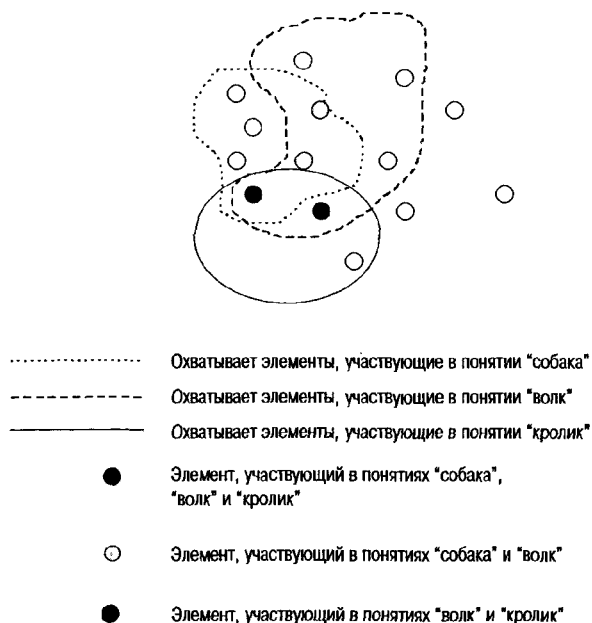


Рис. 8.6. При распределенном представлении элементы участвуют в представлении нескольких понятий, а понятия представляются несколькими элементами

Некоторые представления оказываются не вполне распределенными. Например, для представления понятия можно использовать двоичный вектор, в котором каждый бит означает присутствие или отсутствие определенного признака. Так, для понятия СОБАКА отдельные биты могут представлять признаки типа "имеет хвост", "любит кости" и т.д. При такой форме представления каждое понятие представляется несколькими элементами, но каждый элемент представляет только один признак.

Распределенные представления все же имеют и некоторые недостатки. Предположим, что нейронная сеть обучается принимать решение о том, кому можно дать разрешение на выдачу банковского кредита. Сеть

может быть обучена на таких признаках, как, например, уровень доходов, текущая стоимость имеющихся ценных бумаг, имущественные активы, ежемесячные расходы, длительность работы с банком и т.д. В нашем распоряжении имеется огромная база данных предыдущих займов, и эта база данных включает множество должников, которые не выполнили свои обязательства. Сеть обучается и хорошо работает с большим набором тестовых данных. После этого мы задаемся вопросом: какое знание использует сеть при принятии решения? Информация об этом может оказаться весьма полезной, поскольку мы можем научиться у сети чему-нибудь новому и, что на самом деле более важно, мы сможем больше доверять способности сети выполнить задачу, если выясним, как осуществляется выбор. Чтобы выяснить, какие знания используются для принятия решения, нам необходимо знать, как сеть представляет информацию, а для этого, в свою очередь, нужно знать, что представляют отдельные элементы и весовые значения. Ответить на эти вопросы в случае использования распределенных представлений не всегда просто.

8.3.2. Пространственное сохранение структуры

Обработка структур переменной длины с помощью сети фиксированных размеров является для коннекционистов проблемой нетривиальной. В будущем большинство вычислительных систем на основе нейронных сетей должно иметь модульную архитектуру, где одна сеть должна будет сообщаться с другой сетью, а построение систем на основе интерфейсов, предполагающих обмен данными фиксированных размеров, может упростить задачу. Кроме того, если все представления должны быть одной ширины, то соответствующие понятия можно будет проверять на сходство.

Сеть RAAM отображает символьную структуру в пространственное представление в том смысле, что n значений активности скрытых элементов лежат в n -мерном пространстве. Символьные структуры komponуются с помощью конкатенации составляющих. Простым примером является слово BRAIN, которое komponуется как конкатенация входящих в него отдельных букв. Векторное представление слова BRAIN может быть получено с помощью конкатенации терминальных кодов. Например, используя 5-битовые коды для букв, слово BRAIN можно представить как '10000 01000 00100 00010 00001'. Конкатенация двоичных кодов для построения представлений нейронных сетей оказывается довольно неудобной: словам разной длины будут соответствовать векторы разной длины, если их не дополнять достаточным количеством нулей. При представлении с помощью последовательной конкатенации бинарных деревьев или деревьев более высокой валентности может теряться важная информация, ввиду того, что

при этом утрачивается информация об относительном положении поддеревьев. Например, при обработке деревьев от вершины к основанию и $(V(DN))$, и $((DN)V)$ при конкатенации дадут 'DNV'. Можно избежать этого, указав в векторе положение каждой вершины дерева, но этот метод оказывается довольно громоздким. Сеть RAAM комбинирует представления функциональным методом. Кодер можно интерпретировать как рекурсивно вызываемую функцию. Формирование слова BRAIN тогда будет выглядеть, как $f(N(f(I(f(A(f(R(f(B(NIL)))))))$), где $f(x)$ обозначает кодирование в сети RAAM. При использовании функционального вида отображения поддерева и все дерево в целом накладываются одно на другое.

Структура деревьев на рис. 8.1 управляется правилами грамматики выражений. Эти правила дают явные знания, указывающие допустимые структуры выражений и их взаимные связи, но эти знания становятся менее явными (или неявными) в случае деревьев. Например, если вас просят разбить деревья на группы, вы можете использовать для этого разные критерии, такие как длина дерева, число общих поддеревьев или число терминальных символов как меру длины дерева и т.д. Задача группирования не тривиальна, но существует еще явное группирование, определяемое правилами грамматики: одни деревья попадают в группу S (sentence — предложение), другие же — в группы NP (noun phrase — именная группа), VP (verb phrase — глагольная группа), PP (preposition phrase — предложная группа) и AP (adjective phrase — группа прилагательного). Одно правило для оценки сходства может сделать задачу группирования тривиальной — примером является группирование слов, основанное на их длине (числе символов). Несколько правил для оценки сходства может порождать проблемы, так как в результате делается попытка параллельно удовлетворить несколько конкурирующих ограничений. Пространственное представление структуры делает задачу группирования по сходству более простой, так как оказывается возможным прибегнуть к группированию, основанному на метрике типа евклидова расстояния (см. главу 3). Однако, для того, чтобы группирование имело смысл, мы должны опираться на пространственное представление, несущее в себе некоторую полезную информацию о структуре. К счастью, сеть RAAM генерирует представления, сохраняющие структурную информацию. На рис. 8.7 изображена иерархическая кластерная диаграмма представлений, генерируемых с использованием деревьев, показанных на рис. 8.1. Эти пространственные представления были получены с помощью сети (S)RAAM (см. [Callan, Palmer-Brown, 1977]), которая является математической моделью сети RAAM, но и обычная сеть RAAM порождает подобное группирование (см. [Pollack, 1990]). Кластерная диаграмма показывает, что почти все деревья (кроме одного) попадают в определенную категорию в соответствии с правилом выражения, которому соответствует дерево.

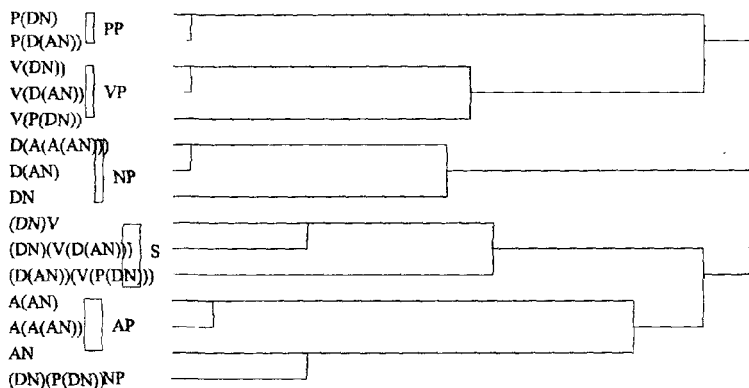


Рис. 8.7. Кластерная диаграмма представлений сети (S)RAAM для деревьев синтаксического анализа

Кластеризация на рис. 8.7 может выглядеть несколько случайной, но в общем сеть RAAM обеспечивает отображение структур в векторы фиксированной длины, в некотором смысле сохраняющее сходство.

Пространственное представление структур интуитивно полезно. Для того чтобы избежать хранения нескольких копий одного и того же знания и обеспечить доказательство по умолчанию в случае отсутствия нужной информации, как инженеры программного обеспечения, так и практикующие специалисты по искусственному интеллекту используют технику наследственности. Наследственность может порождать проблемы в некоторых исключительных ситуациях (например, страус не наследует признак “умеет летать” от класса птиц) и неоднозначность, когда несколько родительских элементов имеют одно и то же свойство (от какого из родителей тогда наследуется соответствующее свойство?). Для разрешения таких проблемных ситуаций можно вводить дополнительные правила, но по определенным причинам (сходным с проблемами выяснения подобия) эти правила могут оказаться чрезмерно ограничительными из-за символической природы формализма моделей, использующих наследственность. Попытки преодолеть проблемы в сетях RAAM, связанные с исключительными ситуациями и неоднозначностью, предпринимались в [Boden, 1996]. Предложенный в указанной работе подход опирается на пространственные представления ассоциаций объектов.

Структура передается сети RAAM путем согласования времени предъявления поддеревьев входному слою. Можно было бы возразить, что

структуры должны возникать каким-то более естественным образом. Например, при восприятии речи говорящего наши органы чувств выделяют последовательности слов, не имеющие явных ярлыков структуры выражения или категоризации слов. В [Elman, 1990] было продемонстрировано, что иерархическая категоризация может возникать из последовательности. В одном из своих экспериментов Элман (Elman) использовал простую рекуррентную сеть (сеть SRN, см. главу 5), с помощью которой из порядка слов должны были выявляться лексические классы. Для создания 10000 двух- и трехсловных фреймов предложений использовались шаблоны. Пример фрейма, показанный на рис. 8.8, генерирует предложения из трех слов — например, “man see woman” (т.е. “мужчина видит женщину”). Каждое слово кодировалось как 31-битовый вектор и имелось 150 контекстных элементов. Все рассматривавшиеся 27534 слова, содержащиеся в 10000 предложениях, были связаны конкатенацией, в результате чего был создан один (27534×31) -битовый вектор. Каждое слово подавалось во входной слой сети SRN вместе с предыдущим контекстом, а задачей сети было предсказание следующего слова последовательности. Сеть обучалась в течение шести полных циклов последовательности, после чего весовые значения были заморожены (т.е. никакого обучения больше не проводилось). Последовательность была пропущена через сеть еще один раз, и все значения активности скрытых элементов были сохранены. Каждое слово последовательности плюс контекст (27534 единиц) были представлены 150-мерными векторами. И хотя на самом деле имелось только 29 уникальных слов, ожидалось, что все 27534 вектора окажутся уникальными ввиду того, что они комбинировались с контекстной информацией. Суммарный вектор каждого слова вычислялся с помощью всех скрытых векторов, активизированных соответствующим словом (вместе с контекстом). Эти векторы затем были сгруппированы на основе использования иерархической техники кластеризации. Кластеризация показала пространственное разделение глаголов и существительных, а также выявила иерархическое размещение категорий слов, как показано на рис. 8.9 и 8.10.

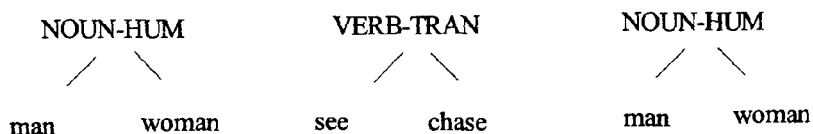


Рис. 8.8. Шаблон для генерирования предложений из трех слов (NOUN означает существительное, а VERB — глагол)

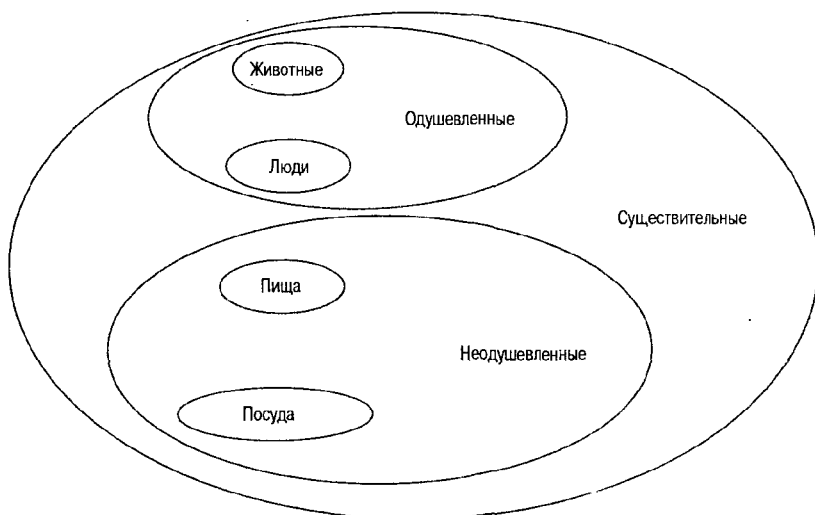


Рис. 8.9. Абстрактная схема кластерной диаграммы. Все представления являются существительными: животные и люди являются одушевленными, пища и посуда — неодушевленными

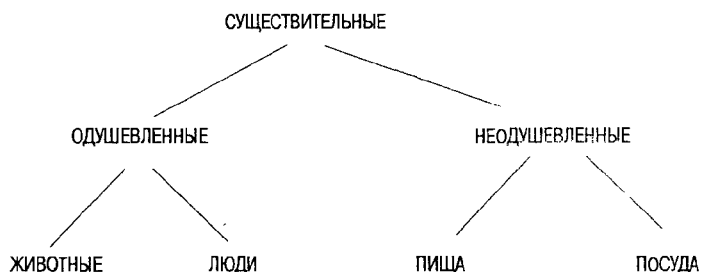


Рис. 8.10. Структура, отображающая пространственное размещение имен существительных, полученная в результате кластеризации значений активности скрытых элементов простой рекуррентной сети. Иерархия является неявной в том смысле, что каждый уровень этой иерархии определяется пространственной близостью

Здесь существует опасность слишком увлечься сохранением структур в пространственном представлении: подобные структуры в конечном итоге размещаются в расположенных рядом областях пространства. Ранее уже было отмечено, что выяснение подобия символьных структур может оказаться непростым делом. Но задача становится совсем сложной, когда

для оценки сходства вообще нет критериев. Например, совсем просто увидеть, соответствуют ли два логических утверждения форме “выражение $\wedge$ выражение”. Такие формы ищутся для того, чтобы сказать, будут ли применимы к данному утверждению правила преобразования или правила вывода. В этом смысле выяснение сходства оказывается простым делом, поскольку символьные структуры создаются с помощью конкатенации элементарных структур (атомов). В случае числовых векторов меры сходства оказываются простыми из-за того, что мы можем оценивать сходство по евклидову расстоянию. Однако использование евклидовой метрики скажем для кластеризации, оказывается при оценке сходства структур полезным только тогда, когда вектор, в который отображается структура, сохраняет информацию об этой структуре. Относительное пространственное размещение представлений сети RAAM может, например, существенно измениться при использовании для терминальных символов случайных векторов. Поэтому обычно терминальным символам назначаются ортогональные векторы битов. Вопросы, рассмотренные в данном разделе, подробно рассматривались в [Sharkey, Jackson, 1995]. Мы здесь можем только отметить, что обычно в сетях типа RAAM имеется возможность получить представления, которые в результате их кластеризации проявят определенные характеристики данных (например, группируя их как существительные, глаголы и т.д. или группируя их по значениям).

8.3.3. Контекст

Нейронные сети, распределяющие и накладывающие представления, обеспечивают контекст — свойство, которое многие коннекционисты провозглашают главным и наиболее важным преимуществом коннекционизма. Предположим, что вам никогда не встречалось словосочетание “палочки для еды”, и в некотором тексте вам попадаетея на глаза предложение “Им подали шпинат с палочками для еды”. Вы не знаете, что такое “палочки для еды”, но на этой стадии вы можете предположить, что это или что-то съедобное, или, возможно, принадлежности для приема пищи. Позже вы читаете: “Палочки для еды вместе с остальными кухонными приборами поместили в посудомоечную машину”. В результате ваша интерпретация “палочек для еды” переместится от возможной интерпретации их как вида овощей к более вероятной интерпретации их как принадлежности для приема пищи — такой вывод оказался возможным ввиду того, что вы знаете, что такое “ложка” и “вилка”, а “палочки для еды” оказались в том же контексте. Элман (Elman) продемонстриро-

вал этот тип концептуального вывода в эксперименте, описанном выше. Когда во всех 10000 предложениях Элман заменил слово “man” (человек) словом “zog” (отсутствующим в словаре) и подал эти видоизмененные предложения на рассмотрение ранее обученной сети, слово “zog” показало те же пространственные связи, что и слово “man”.

Рассматриваемый нами тип нейронной сети порождает разные представления для одного и того же слова, появляющегося в разном контексте. Например, “лимонад” в выражении “баночный лимонад” будет иметь представление, отличное от его представления в выражении “бутылочный лимонад”. Это кажется неудобным, поскольку само слово в символическом печатном виде выступает как фиксированный образ, не зависящий от контекста (если не принимать во внимание различия в шрифте или размерах). Контекстная зависимость представлений ставит перед коннекционистами трудные проблемы при попытках выполнения вычислений определенного типа (например, при реализации структурно-сенситивных процессов; см. также примеры, о которых речь пойдет ниже). Однако это свойство кажется полезным. Возвращаясь к примеру с бутылкой/банкой лимонада, заметим, что хотя “лимонад” и остается одной и той же прозрачной жидкостью и в бутылке, и в банке, некоторые знатоки могут замечать разницу во вкусе в зависимости от того, в какую упаковку эта жидкость разлита. Мы должны ожидать, что представления “понятий лимонада” окажутся очень близкими в семантическом пространстве, но не в точности одинаковыми, чтобы разница в зависимости от контекста оказалась вполне разумной.

В [Kohonen, 1990] было продемонстрировано генерирование семантической карты слов с помощью самоорганизующейся карты признаков (см. главу 3). С помощью метода, в некотором смысле подобного методу Элмана, Кохонен с помощью шаблона получил набор трехсловных предложений (см. раздел 8.3.2). Предложения объединялись конкатенацией, и контекст для каждого слова определялся с помощью усреднения по всем словам, непосредственно предшествующим данному слову и следующим за ним. Вектор каждого слова получался в результате конкатенации кода вектора слова и ближайшего контекстного вектора (некоторые детали этого процесса здесь опускаются). Затем вектор-результат использовался для обучения самоорганизующейся карты признаков. Получаемая семантическая карта показала разделение слов на существительные, глаголы и наречия, а также дальнейшую семантическую организацию внутри этих регионов. Кохонен отмечает: “Мы ясно видим, что контекст “заставляет” словарные элементы запоминать позиции, в которых их размещение отражает как грамматические, так и семантические связи”.

8.3.4. Символьное представление и представление нейронных сетей

Символы произвольны в том смысле, что один символ может быть заменен другим. Если мы в некотором тексте слово ТАБЛИЦА заменим словом АЦИЛБАТ, читатель может счесть текст немного неудобным для восприятия, но все еще доступным для понимания. Символ ТАБЛИЦА ссылается на объекты, опыт работы с которыми читатель имеет, и простое изменение метки на АЦИЛБАТ на этот опыт не повлияет. Символы являются просто метками и воспринимаются как представления только тогда, когда используются наблюдателем, имеющим приобретенные (на основе опыта) знания о том, на что ссылается конкретный символ. Другими словами, символы есть ничто без их творцов. Символы являются дискретными сущностями и то, на что они ссылаются, иногда зависит от имеющегося рядом текста. Например, “уход” в предложениях “Уход за новорожденным” (забота) и “Уход артиста со сцены” (определенное действие) является дискретным символом, но решение о том, какому из понятий этот символ соответствует, принимаем мы, в зависимости от связей с расположенным рядом текстом.

В противоположность этому, представления нейронных сетей не произвольны, а являются контекстно-зависимыми. Это интуитивно полезное свойство ставит трудный вопрос: сколько контекстов необходимо рассмотреть для того, чтобы представление оказалось универсально полезным (например, для общего понимания разговорной речи)? Можно привести пример смысловой разницы между произвольным символьным представлением и представлением, полученным через внедренный контекст. Представьте себе группу девушек-моделей, работающих в некотором модельном агентстве. Дом моды уже пользовался услугами всех этих моделей в прошлом, поэтому главный дизайнер знает их всех по именам. Главный дизайнер встречается с представителем агентства, чтобы обсудить то, кто из моделей будет участвовать в предстоящем показе мод. В ходе встречи на любую из моделей можно ссылаться по имени или по фотографии. Как имя, так и фотография являются замещением реального объекта, но фотография имеет более полезное содержание, поскольку предлагаемая ею форма информации зависит от реального представляемого объекта.

Конечно, символьные структуры являются композиционными, и соответствующие процессы могут быть структурно-сенситивными. Трудно вообще найти вычислительные системы, не обладающие такими свойствами. В результате композиционной природы структур имеется возможность разбить структуру на составные части (например, для модульной обработки), а структурно-зависимая обработка дает высокую степень

обобщения. Предположим, что требуется создать символ BACKACHE (боль в спине) из символов BACK (спина) и ACHE (боль). Если символы рассматривать как строки (наши вычислительные ярлыки), то мы знаем, что новый символ BACKACHE является просто конкатенацией BACK и ACHE, а на нижнем уровне функция *конкатенации* будет знать, что BACK имеет четыре символа, и в конкатенации новая строка будет содержать ACHE начиная с пятой символической позиции. В рамках символизма, представляющие ярлыки, такие как строки, соответствуют вычислительным ярлыкам, допускающим их использование в явном алгоритме решения требуемой задачи (в данном случае это просто конкатенация). В случае нейронной сети вычислительные ярлыки являются значениями активности и значениями весов элемента. Алгоритмы обучения вычисляют ввод и вывод элементов и обновляют весовые значения. Если мы должны использовать сеть RAAM с 25 скрытыми элементами для представления символов BACK и ACHE, то представления не будут лежать на уровне вычислительных ярлыков. В сети RAAM все 25 элементов используются и для того, чтобы представить BACK, и для того, чтобы представить ACHE, поэтому то, как создать BACKACHE, будет совсем не очевидно. Мы не можем указать на один из 25 элементов и сказать, чему явно соответствует этот элемент (например, В или С и т.д.). Поэтому мы не можем взять 25 элементов сети RAAM и указать, какое действие и с каким из элементов необходимо выполнить для того, чтобы получить конкатенацию. Чтобы выполнить конкатенацию, можно прибегнуть к тому, что называют *целостным* подходом. Можно, например, обучить другую сеть с прямой связью, которая получила бы на вход представление RAAM для BACK и ACHE и произвела на выходе представление RAAM для BACKACHE. Можно обучить такую сеть на сотнях примеров в надежде получить сеть, выполняющую функцию *конкатенации*. Сеть *конкатенации* может правильно выполнять обобщение при ее тестировании на новых примерах, но наша уверенность в том, что эта сеть правильно выполнит *конкатенацию* любой пары символов, будет неполной из-за того, что мы не знаем точного механизма, который использует при этом сеть. Главное преимущество символизма заключается в том, что если некоторая функция (например, *конкатенация*) была однажды правильно определена как действующая на определенные типы вычислительных ярлыков, то в дальнейшем мы можем быть уверенными в том, что, не принимая во внимание возможность ошибки человека и отказ аппаратных средств, данная функция будет правильно работать с любыми ярлыками, лишь бы они были соответствующего типа.

Хотя нейронные сети все еще не выдерживают критики как символичные процессоры для задач типа конкатенации, существует ряд примеров

демонстрации целостного подхода, не упоминавшихся в предыдущем параграфе, но обеспечивающих в какой-то мере ответ на критику, утверждающую, что коннекциям не присуща систематичность. Вспомните, что для того, чтобы некоторая система была систематичной, она должна вести себя так, как будто она понимает “Джон любит Мери”, если она понимает “Мери любит Джона” или если она понимает “Джейн думает, что Мери любит Джона”. Символьную систему можно заставить вести себя систематическим образом по различным сценариям. Например, если в составе некоторой структуры система содержит выражение любит(X, Y), то ей можно задать общие вопросы об отношении любви (“любит” является предикатом, который следует при этом искать, а аргументы являются переменными, которые могут обозначать любую константу). В [Niklasson, Sharkey, 1997] предлагается пример структурно-зависимой обработки структур системой нейронной сети. Было показано, что с помощью нейронной сети можно выполнять преобразования логических утверждений. Например, $(P \wedge Q) \Rightarrow R$ можно преобразовать в его логический эквивалент $\neg(P \wedge Q) \vee R$.

Троичная сеть RAAM (т.е. сеть с тремя входными полями) с топологией 30-10-30 использовалась для представления 156 формул с высказываниями разной сложности. После завершения обучения сети RAAM половина из 156 построенных сетью RAAM представлений использовалось для того, чтобы обучить преобразующую сеть выполнению преобразований следующего типа:

(выражение $\Rightarrow$ выражение $\Leftrightarrow \neg$ выражение $\vee$ выражение).

Например, вводом преобразующей сети могло быть представление RAAM для $(P \wedge Q) \Rightarrow R$, а целевым выводом — представление RAAM для $\neg(P \wedge Q) \vee R$. Преобразующая сеть представляла собой стандартную гетероассоциативную сеть с прямой связью (см. главу 4). Преобразованное представление RAAM декодировалось к его составляющим с помощью декодера RAAM. Терминальным символам соответствовали двоичные векторы с ярлыками типа, как показано в табл. 8.3. Так, $\wedge$ и $\vee$ имеют один тип, но, в отличие от правил грамматики предложений из главы 7, $\Rightarrow$ и $\neg$ не классифицируются как связки. Ниже приведены правила, управляющие синтаксисом формул.

формула $\rightarrow$ [выражение $\Rightarrow$ выражение]

формула $\rightarrow$ [$\neg$ выражение $\vee$ выражение]

выражение $\rightarrow$ [элементарное | сложное]

элементарное $\rightarrow$ [P | Q | R]

сложное $\rightarrow$ [элементарное связка сложное]

связка $\rightarrow$ [$\wedge$ | $\vee$]

В 78 тестовых преобразованиях не были правильно преобразованы 20 формул. Во всех 20 неправильных преобразованиях ошибка содержалась в преобразовании составляющего в составляющий того же типа. Этот уровень обобщения можно считать высоким с учетом того, что преобразующая сеть не обучалась ни на одном из 78 тестовых примеров.

Таблица 8.3. Ортогональные векторы, соответствующие терминальным символам

Терминальный символ	Двоичный код
P	1000 100000
Q	1000 010000
R	1000 001000
∧	0100 100000
∨	0100 010000
⇒	0010 100000
¬	0001 100000
NIL	0000 000000

Можно заподозрить, что успех этого эксперимента зависел от способа присвоения ярлыков типа терминальным символам. Чтобы рассеять такие подозрения, эксперимент был повторен без использования ярлыков типа, а с использованием кодирования всех терминальных символов случайными векторами битов — для кодирования использовались 20 битов, и вероятность того, что бит установлен равным 1, была 3/20. Обобщение оказалось немного лучше, чем в предыдущем эксперименте.

Другой эксперимент из [Niklasson, Sharkey, 1997] показал, что процент обобщения оказывается высоким даже тогда, когда преобразующей сети приходится работать с другими законами преобразования логики. Например,

¬(P ∨ Q) преобразуется в логически эквивалентную форму ¬P ∧ ¬Q,

¬(P ∧ Q) преобразуется в логически эквивалентную форму ¬P ∨ ¬Q.

Хотя в этих экспериментах нейронные сети демонстрируют впечатляющий уровень обобщения, они еще не идут ни в какое сравнение с систематичностью, присущей символьным правилам преобразования. Однако будущее нейронных сетей выглядит многообещающим. Никлассон (Niklasson) и Шарки (Sharkey) смогли продемонстрировать, что нейронная сеть может научиться тому, какие символьные типы являются допустимыми на определенных позициях (а не просто тому, какие допус-

каются символичные ярлыки). Обобщение “типы” в данном случае является важным шагом по направлению к более сильным свойствам систематичности. Например, слабая с точки зрения систематичности система может получить обобщение любит(джон, джейн) по набору экземпляров {любит(джон, мери), любит(дэвид, мери), любит(дэвид, джейн)}, но сильная система должна получить обобщение любит(Человек, Человек).

Коннекционисты уже начали отвечать на критику, но мы не должны забывать и о преимуществах нейронных сетей в вопросах обеспечения плавного снижения эффективности, быстрой параллельной обработки и чувствительности к контексту. Мы должны помнить и о том, что представления нейронных сетей не являются произвольной конкретизацией, а являются чувствительными к вводу (т.е. к тому, что они принимают из внешней среды).

8.4. Обработка речи

8.4.1. Синтаксический анализ

В [Reilly, 1992] описан пример синтаксического анализа предложений. Задачей используемого там анализатора является выяснение категории каждого слова в отдельности (существительное, глагол и т.д.) и построение дерева синтаксического анализа всего предложения. Такое динамическое построение дерева синтаксического анализа из последовательностей называется *оперативным синтаксическим анализом*. Например, категориями слов, составляющих предложение “The boy kicked the ball” (“Мальчик ударил по мячу”), являются D N V D N, а деревом синтаксического анализа— (D N)(V (D N)). Синтаксический анализатор, о котором идет речь, представляет собой простую рекуррентную сеть (сеть SRN), при обучении которой целевым выводом является представление RAAM целого дерева синтаксического анализа. Шаги, выполняемые при синтаксическом анализе отдельного предложения, показаны на рис. 8.11.

Райли (Reilly) показал, что можно построить оперативный синтаксический анализатор, используя сети RAAM и SRN. При этом обнаружились проблемы при достижении хороших свойств обобщения системы, да и само обучение тоже отчасти вызывало трудности. Однако, как и в случае любой нейронной модели, свойства обобщения могут быть, скорее всего, улучшены путем изменения режима обучения. Главной целью Райли была проверка возможности проведения оперативного синтаксического анализа, поэтому учебное множество было совсем небольшим (16 последовательностей).

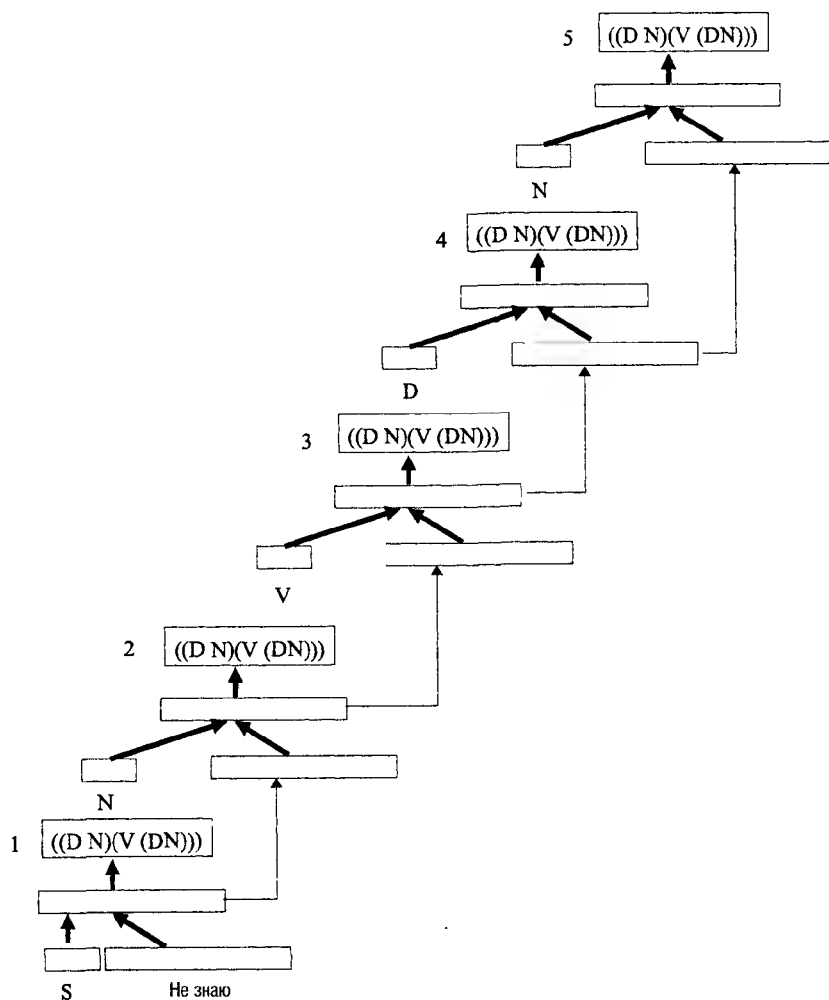


Рис. 8.11. Пять шагов, выполняемых при синтаксическом анализе предложения типа "The boy kicked the ball" с помощью простой рекуррентной сети. Входными данными являются категории слов, а выводом – представление RAAM дерева синтаксического анализа

В [Sharkey, Sharkey, 1992] тоже был построен синтаксический анализатор, использующий сети RAAM и SRN. В модели этих авторов сеть SRN обучалась предсказанию следующего слова в последовательности слов. Сеть с прямой связью и тремя слоями обучалась отображению

представлений скрытого слоя элементов сети SRN в представления сети RAAM деревьев синтаксического анализа. Во время выполнения предложение читалось сетью SRN, чтобы сгенерировать для него представление скрытого слоя, а затем это представление отображалось в представление RAAM, которое в конце концов декодировалось, чтобы выяснить истинную структуру выражения.

8.4.2. Преобразование предложений

Разговорный язык очень выразителен и позволяет передать одну и ту же мысль несколькими способами. Одно и то же сообщение можно передать также в активной или пассивной форме. Например, пассивной формой предложения “Джон построил дом” является “Дом построен Джоном”. Челмерс показал (см. [Chalmers, 1990]), что система на основе нейронной сети позволяет осуществлять преобразование активной формы в пассивную. Он тоже применил подход, использованный Никлассоном и Шарки для преобразования логических утверждений (см. раздел 8.3.4): сеть RAAM использовалась для представления предложений в обеих формах, а сеть с обратным распространением ошибок обучалась преобразованию активной формы предложения в пассивную. Было сгенерировано 125 предложений, и 40 из них было выбрано для обучения сети RAAM. Сеть RAAM была обучена представлять как активную, так и пассивную форму этих 40 предложений (т.е. в результате было получено 80 представлений предложений). Для проверки свойств обобщения сети было закодировано другое множество из 40 предложений (80 предложений с учетом активной и пассивной форм каждого из них), которые затем декодировались. Число закодированных тестовых представлений, которые не декодировались правильно, оказалось равным 13. Преобразующая сеть обучалась на тех же 80 предложениях, которые использовались для обучения сети RAAM. Представления сети RAAM для 40 предложений в активной форме служили вводом, а целевым выходом были соответствующие 40 представлений сети RAAM в пассивной форме. По окончании обучения все представления пассивных форм предложений, полученные на выходе преобразующей сети, могли быть декодированы сетью RAAM. Обобщение проверялось на 40 предложениях, используемых для тестирования сети RAAM. Из 40 сгенерированных представлений пассивной формы предложений, 26 были правильно декодированы сетью RAAM.

И хотя Челмерс отмечает, что показатель обобщения оказался лучше, чем он ожидал, он пожелал выяснить, где следует искать причину неспособности сети выполнить обобщение на некоторых структурах в сети RAAM или преобразующей сети. Выше уже отмечалось, что сеть RAAM

оказалась не способной дать 100% обобщения. Чтобы исключить возможность появления ошибки обобщения из-за сети RAAM, для обучения RAAM использовались все 125 (или 250 с учетом активной и пассивной форм) предложений. Затем преобразующая сеть обучалась на 75 из 125 пар активно-пассивных форм предложений. Показатель обобщения оказался равным 100% при тестировании преобразующей сети на других 50 парах активно-пассивных форм.

Челмерс также продемонстрировал, что другая преобразующая сеть тоже достигала 100% показателя обобщения при обучении преобразованию предложений из пассивной формы в активную.

Эксперимент Челмерса еще раз показал, что система на основе нейронной сети может выполнять структурно-зависимые преобразования. В [Callan, Palmer-Brown, 1997] эксперимент Челмерса был повторен с использованием архитектуры (S)RAAM. Показатель обобщения преобразующей сети тоже оказался равным 100%, но, в отличие от сети RAAM Челмерса, сеть (S)RAAM не требует обучения на всех предложениях. Этот результат весьма интересен. Сеть (S)RAAM не обучается традиционными методами типа метода обратного распространения ошибок. Обучение при условии, что представления сети (S)RAAM имеют больше ограничений, чем представления RAAM (например, число скрытых элементов оказывается больше), и тот факт, что сеть (S)RAAM “хорошо выполняет обобщения”, говорят о том, что обучающее множество является очень разнообразным. Хотя обобщение сети RAAM в эксперименте Челмерса было хорошим, оно оказалось все же несовершенным, а результат эксперимента с сетью (S)RAAM указывает на то, что вполне возможно достичь лучшего обобщения и при использовании сети RAAM. Как правило, весь потенциал качества работы сети реализовать очень трудно.

8.4.3. Завершенные модели

В этом подразделе мы рассмотрим две системы на основе нейронных сетей, которые пытаются выполнить задачу обработки живого языка в более полном объеме. Обе системы опираются на идею использования сценариев.

Предполагается, что сценарии должны отражать стереотипные ситуации. С помощью сценария можно восполнить отсутствие некоторой информации в текстах, подобных следующему.

Джулиан пошел с друзьями в их любимый индийский ресторан. Они заказали еду и много воды, все съели и не забыли оставить чаевые перед тем, как уйти.

Из этого ограниченного текста, имея опыт посещения индийских ресторанов, можно ответить на вопросы, подобные следующим.

Почему они заказали много воды?

Кто подавал пищу?

Кому они оставили чаевые?

В большинстве случаев посещение ресторана складывается из ряда стандартных действий: вход в ресторан, размещение посетителя за столом официантом, выбор блюд из меню, принятие заказа, прием пищи, оплата по счету и т.д. Сценарий содержит роли. Роли могут предназначаться для клиента, официанта, шеф-повара и т.д. Так, в сценарии, приведенном выше, одна из ролей уже занята (Джулиан играет роль клиента).

Качество работы двух рассматриваемых моделей будет обсуждаться после того, как они будут описаны.

Обработка сценариев в модели DYNASTY

Модель DYNASTY (DYNAmic STory understanding system — Динамическая система понимания текста), предложенная в [Lee *et al.*, 1990], является модульной системой на основе нейронных сетей, получающей на входе основанный на сценариях фрагмент текста и дающей на выходе заверченный пересказ (парафразу). Давайте используем перевод примера создателей модели DYNASTY и рассмотрим следующий ввод.

Джон зашел в Чарт-Хаус. Джон съел бифштекс. Джон оставил чаевые.

Такой ввод порождает следующий вывод.

Джон зашел в Чарт-Хаус. Официант усадил Джона. Официант принес меню. Джон прочитал меню. Джон заказал бифштекс. Джон съел бифштекс. Джон заплатил по счету. Джон оставил чаевые. Джон вышел из Чарт-Хауса и направился домой.

Действие в модели DYNASTY концентрируется на распределенных семантических представлениях, соответствующих понятиям (например, молоко и человек) и утверждениям (например, человек пьет молоко через соломинку). Распределенные семантические представления генерируются с помощью расширений сети RAAM (XRAAM — extended RAAM), являющихся по сути сетями RAAM с глобальным словарем, хранящим пары “символ”—“распределенное семантическое представление”. Например, символ “молоко” должен быть сохранен вместе с его представлением RAAM. Для выполнения этой задачи используется целый ряд взаимодействующих модулей.

Распределенное семантическое представление для слова-понятия строится таким образом, чтобы представление отражало что-то вроде всех различных ролей, в которых это слово используется. Например, роль слова *молоко* в предложении “человек пьет молоко” немного отличается от роли того же слова в “человек ест хлеб с молоком”: в первом случае мо-

локо является главным объектом, а во втором *молоко* является сообъектом. При создании представления понятия *молоко* свой вклад внесет каждое из предложений, в которых встречается слово молоко. С каждым предложением связывается некоторая структура. Например, шаблон предложения

человек ест пищу с помощью прибора

имеет структуру

АГЕНТ-ДЕЙСТВИЕ-ОБЪЕКТ-ИНСТРУМЕНТ.

Экземпляр “мужчина ест спагетти с помощью вилки” соответствует конкретизации ролей [АГЕНТ-мужчина, ДЕЙСТВИЕ-ест, ОБЪЕКТ-спагетти, ИНСТРУМЕНТ-вилка]. Имеется двухсторонняя связь между словами-понятиями и предложениями в том смысле, что семантическое содержание предложения зависит от каждого из его слов-понятий, и каждое слово-понятие зависит от каждого из предложений, в которых оно встречается. В определенной мере эта зависимость учитывается в процессе обучения созданию распределенных семантических представлений. Чтобы выучить слова-понятия, используется кодирующая понятия сеть, а чтобы выучить предложения, используется сеть, кодирующая предложения. Обе эти сети являются сетями RAAM, связанными с глобальным словарем. Обучение обеих сетей RAAM взаимозависимо в том смысле, что на вход сети кодирования понятий подаются представления, создаваемые сетью кодирования предложений, и наоборот. В ходе обучения распределенным семантическим представлениям кодируются все слова-понятия, затем кодируются все предложения, и процесс повторяется до тех пор, пока не будут найдены устойчивые образцы распределенных семантических представлений (т.е. такие, для которых не будет значительных изменений значений активности скрытых элементов обеих сетей). Слова-понятия и предложения подаются на рассмотрение сети в виде троек. Слово-понятие структурируется как “слово-понятие, конкретизация-роль, предложение-метка”, а предложение — как “предложение-метка, конкретизация-роль, слово-понятие”. Используется троичная сеть RAAM. Например, предположим, что изучается слово-понятие *молоко* и что *молоко* встречается в четырех предложениях.

p1. Мужчина пьет молоко через соломинку.

p2. Компания доставляет молоко в картонной упаковке.

p3. Люди получают молоко от коров.

p4. Мужчина ест хлеб с молоком.

Структурными тройками слов-понятий для сети являются

(молоко ОБЪЕКТ p1)

(молоко ОБЪЕКТ p2)

(молоко ОБЪЕКТ p3)

(молоко СООБЪЕКТ p4),

а структурными тройками предложения p1 являются

(p1 ДЕЙСТВИЕ пьет)

(p1 ОБЪЕКТ молоко)

(p1 ИНСТРУМЕНТ соломинка)

(p1 АГЕНТ мужчина)

Древоподобные структуры, показанные на рис. 8.12 и 8.13, должны помочь разобраться, каким образом происходило обучение сетей RAAM.

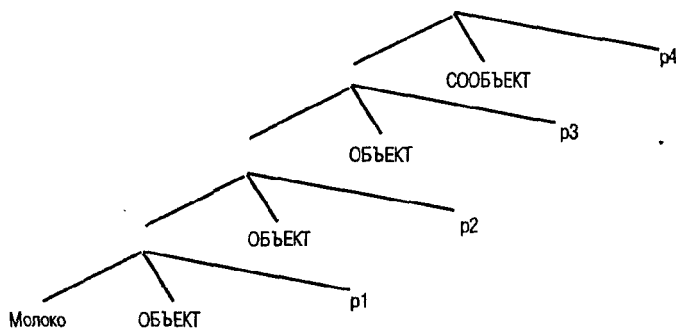


Рис. 8.12. Полная древоподобная структура кодирования слова "молоко" в примере сети RAAM. Слово "молоко" встречается в четырех предложениях

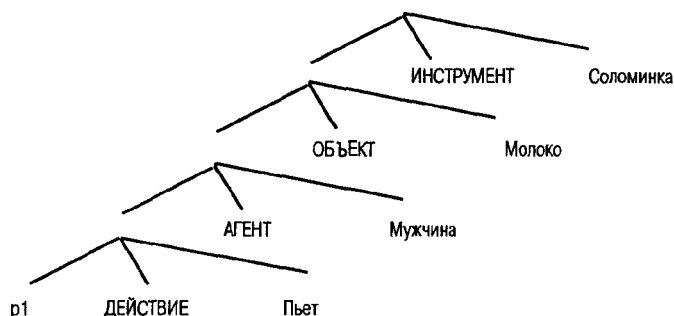


Рис. 8.13. Полная древоподобная структура кодирования предложения "Мужчина пьет молоко через соломинку"

Обучение распределенным семантическим представлениям в обеих сетях происходит одновременно. В начале обучения сжатых представлений для $p1$ и “молоко” не существует, поэтому их образцы установлены на “не знаю” (все значения элементов поля I изначально установлены равными 0.5). Все конкретизации ролей (для АГЕНТ, ОБЪЕКТ и т.д.) фиксируются на ортогональных образцах.

Модель DYNASTY содержит целый ряд следующих модулей.

1. Обучающий модуль распределенного семантического представления, состоящий из двух модулей XRAAM, описанных выше.
2. Кодер событий, являющийся троичной сетью RAAM.
3. Опознаватель сценариев, представляющий собой рекуррентную сеть, задачей которой является распознавание типа сценария по последовательности событий.
4. Основной генератор, представляющий собой рекуррентную сеть, задачей которой является генерирование полной последовательности событий.

Кодер событий использует троичную сеть RAAM для представления троек вида “событие, конкретизация-роль, слово-понятие”. На рис. 8.14 показана схема структуры ввода для “Джон зашел в Чарт-Хаус”. Предложения и события являются, по существу, одним и тем же, но сети, формирующие представления для предложений и событий, различны. Напомним, что предложения представляются в обучающем модуле распределенного семантического представления, а кодер событий использует распределенные семантические представления.

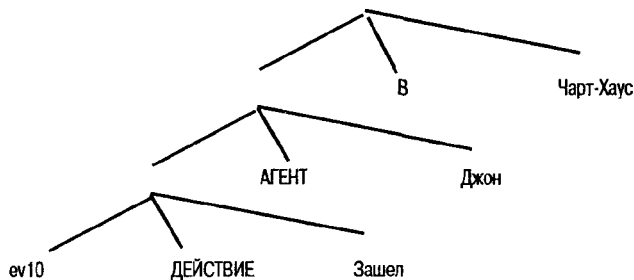


Рис. 8.14. Структура кодирования события “Джон зашел в Чарт-Хаус” в сети RAAM. Символ “ev” с номером является просто ярлыком

Ролями ресторанного сценария являются следующие.

Роли сценария: КЛИЕНТ, НАЗВАНИЕ-РЕСТОРАНА, ПИЩА

Экземпляры: Джон, Джек, Чарт-Хаус, Корейский-Сад, бифштекс, ребрышки

КЛИЕНТ зашел в НАЗВАНИЕ-РЕСТОРАНА

официант усадил КЛИЕНТ

официант принес меню

КЛИЕНТ прочитал меню

КЛИЕНТ заказал ПИЩА

КЛИЕНТ съел ПИЩА

КЛИЕНТ оплатил счет

КЛИЕНТ оставил чаевые

КЛИЕНТ покинул НАЗВАНИЕ-РЕСТОРАНА и направился домой

Если вместо КЛИЕНТ использовать Джон, вместо НАЗВАНИЕ-РЕСТОРАНА — Чарт-Хаус, а вместо ПИЩА — бифштекс, мы получим пример сценария, приводившийся выше. Первой стадией обучения является определение распределенных семантических представлений для всех ролей сценария, используемых в обучающих сценариях экземпляров и других понятий (меню, зашел и т.д.). Полным набором ролей будет {ДЕЙСТВИЕ, АГЕНТ, ОБЪЕКТ, СООБЪЕКТ, ИНСТРУМЕНТ, ИЗ, В, МЕСТО, ВРЕМЯ}. Так, для события

КЛИЕНТ зашел в НАЗВАНИЕ-РЕСТОРАНА

мы имеем

ДЕЙСТВИЕ-зашел, АГЕНТ-КЛИЕНТ, В-НАЗВАНИЕ-РЕСТОРАНА.

Точно также, для

Джон зашел в Чарт-Хаус

имеем

ДЕЙСТВИЕ-зашел, АГЕНТ-Джон, В-Чарт-Хаус.

Чтобы получить правильную парафразу (т.е. полный список событий из некоторого ограниченного подмножества), модель DYNASTY должна связать заполнители с ролями сценария (например, связать Джон и КЛИЕНТ). Процедура связывания ролей будет рассмотрена сразу же после краткого обсуждения опознавателя сценариев и основного генератора.

Задачей опознавателя сценариев является распознавание сценариев (ресторан, поход за покупками и т.д.) по последовательности событий. Использование соответствующей сети подобно сети SRN, поскольку значения активности скрытого слоя подаются обратно во входной слой для того, чтобы служить контекстом. Обучение происходит по методу обратного рас-

пространения ошибок, а учебные данные включают несколько экземпляров различных сценариев. Типы сценария определяются ортогональными образцами, и целевым выводом является соответствующий тип сценария.

Задачей модуля основного генератора является создание полной парафразы (т.е. всех событий) для конкретного типа сценария. Поэтому вводом сети основного генератора является тип сценария, а выводом — полный список событий в форме ролей сценария (без каких бы то ни было связываний). Например, первыми двумя событиями для сценария ресторана являются “КЛИЕНТ зашел в НАЗВАНИЕ-РЕСТОРАНА” и “официант усадил КЛИЕНТ”. Сеть обучена генерировать события, представленные кодером событий, и имеет архитектуру, подобную модулю распознавания сценариев. Одно поле элементов входного слоя предназначено для того, чтобы выступать в качестве контекста, а другое поле элементов получает информацию о типе сценария.

В модели DYNASTY процесс получения полной парафразы из некоторого подмножества текста выглядит следующим образом.

1. Входной текст представляется в форме троек событий.
2. Для каждого символа в тройке событий ищется соответствующий образец распределенного семантического представления из глобального словаря.
3. Используется кодер событий, чтобы создать представления событий.
4. Используется опознаватель сценариев, чтобы выявить тип сценария.
5. С помощью основного генератора порождается полное множество событий.
6. Каждое событие шага 5 декодируется в форму тройки-события (с помощью сети кодера событий, т.е. декодера RAAM). Этот процесс разделяет представление события на конкретизации ролей и распределенные семантические представления, определяющие данное событие.
7. Выполняется связывание ролей сценария (описывается ниже).
8. В глобальном словаре ищутся распределенные семантические представления и выбирается связанный с ними символ.
9. Создается полная парафраза.

Заданием редактора связей ролей является связывание ролей сценария с экземплярами. Например, ввод в системе DYNASTY может быть следующим:

Джон зашел в Чарт-Хаус.

Джон съел бифштекс.

Джон оставил чаевые.

Первой вводимой тройкой событий является (ev10 ДЕЙСТВИЕ зашел), (ev10 АГЕНТ Джон) и (ev10 В Чарт-Хаус). Первой тройкой событий от основного генератора будет (ev1 ДЕЙСТВИЕ зашел), (ev1 АГЕНТ КЛИЕНТ) и (ev1 В НАЗВАНИЕ-РЕСТОРАНА). Для ввода ДЕЙСТВИЕ имеет тот же заполнитель (зашел), что и в первой выходной тройке событий основного генератора, поэтому эти роли сценария выбираются и сохраняются в таблице связей. В данном примере две роли (КЛИЕНТ, Джон) и (НАЗВАНИЕ-РЕСТОРАНА, Чарт-Хаус) сохраняются. Затем следующая входная тройка событий сравнивается со следующей порожденной основным генератором тройкой событий. Если заполнители для ДЕЙСТВИЕ не совпадают, для сравнения используется следующая тройка событий, порожденная основным генератором. Процесс повторяется до тех пор, пока не будут обработаны все события основного генератора. Затем замещаются роли сценария во всех выходных событиях.

Модель DISCERN

Модель DISCERN (Distributed SScript processing and Episodic memoRy Network — сеть обработки распределенных сценариев и эпизодической памяти, см. [Mikkulainen, 1994]) на сегодня является одной из наиболее полных систем понимания речи. Подобно модели DYNASTY, здесь тоже все строится на основе сценариев. Расширенная парафраза оригинального рассказа может быть получена из ограниченного текстового ввода, а кроме того, DISCERN может отвечать на вопросы (например, “Что ел Джон в Ма-Мейзон?”). Вводом и выводом DISCERN являются распределенные представления лексических слов (т.е. словесных символов). Модель DISCERN состоит из нескольких модулей.

Анализатор

Анализатор предложений читает слова по одному и строит представление для каждого предложения. Анализатор рассказов создает представление сюжета из последовательности предложений и сохраняет это представление в памяти.

Генератор

Генератор рассказов, используя сохраненное представление, создает полную парафразу, генератор предложений выводит все соответствующие слова.

Память

Словарь реализован в виде двух карт признаков (см. главу 3). Первая (лексическая) карта хранит распределенное представление символьной формы (такой как, например, СОБАКА), а вторая (семантическая) карта — распределенное представление семантики слова. Распределенные

структуры лексикона отражают визуальные характеристики слова, поэтому слова ЛАК и ПАК будут храниться рядом. Распределенные структуры семантической карты отражают характеристики использования слов, поэтому слова ЖЕРТВА и ХИЩНИК также будут храниться рядом. Эпизодическая память хранит сценарий, вариант (например, сценарий ресторана имеет варианты изысканной кухни и быстрого питания) и ролевые связи.

Экземпляры рассказов хранятся в виде иерархических карт признаков. Сценарии сохраняются в одной карте, которая связывается с картой вариантов, а последняя, в свою очередь, связывается с картой признаков ролевых связей. Сжатое представление рассказа может использоваться для подсказки карте сценария, распространяющей соответствующую активность, в результате чего восстанавливается полная парафраза.

Ответы на вопросы

Анализатор формирует представление вопроса, чтобы использовать это представление в качестве подсказки для эпизодической памяти. После этого генерируется соответствующее представление рассказа, используемое вместе с вопросом отвечающим модулем для создания представления ответа, а последнее подается затем на вход генератора предложений.

Задачи типа анализа предложений и создания ответа выполняются с помощью иерархически организованных модулей FGREP (Forming Global Representations with Extended backPropagation — формирование глобальных представлений с помощью расширенного алгоритма обратного распространения ошибок; см. [Miikkulainen, Dyer, 1991]). Модули FGREP по сути представляют собой трехслойные сети с обратным распространением ошибок. Данные ввода и вывода этих сетей сохраняются в словаре. Если модуль FGREP обрабатывает последовательность ввода или вывода, на каждой стадии значения активности скрытого слоя сохраняются, а затем подаются обратно скрытому слою вместе со следующим элементом последовательности. Модули формирования подсказок и создания ответа используют нереккуррентные сети FGREP. Анализаторы и генераторы являются рекуррентными, и при этом первые получают последовательный ввод, а последние — генерируют последовательный вывод.

Входные и выходные слои модуля FGREP разбиты на поля, число которых зависит от задачи модуля. Нереккуррентный модуль FGREP можно обучить созданию отображения синтаксических составляющих предложения в конкретизации ролей. Поля входного слоя могут означать синтаксические составляющие {Подлежащее, Сказуемое, Объект, Дополнение}, а поля выходного слоя — ролевые составляющие {Агент, Действие, Объект, Инструмент, Модификатор}. Пример подобного отображения показан на рис. 8.15.

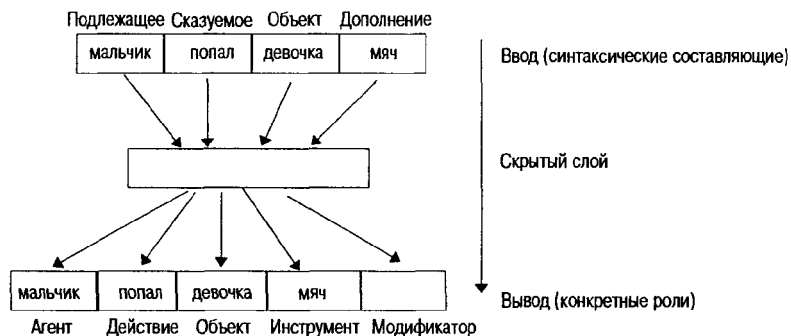


Рис. 8.15. Отображение синтаксических категорий в конкретизации ролей

Поля во входном и выходном слоях загружаются данными из соответствующей слову записи словаря. В процессе обучения записи словаря постоянно меняются. Сначала в словарь заносятся случайные данные, которые в ходе обучения меняются до тех пор, пока к концу обучения они не стабилизируются. Представления изменяются во входном уровне, для чего используется расширенный алгоритм обратного распространения ошибок. В отличие от стандартного алгоритма обратного распространения ошибок, когда должны вычисляться только ошибки для скрытых элементов, чтобы скорректировать первый слой весовых значений, сеть FGREP идет на шаг дальше и вычисляет ошибки для входных элементов. Изменения значений активности каждого входного элемента определяются произведением соответствующего сигнала ошибки и нормы обучения. Сутью метода является идея трактовки входных представлений как дополнительного слоя весов.

Анализатор предложений использует рекуррентную сеть FGREP с одним входным полем (поскольку слова читаются по одному) и с одним выходным полем для каждой конкретизации роли. Анализатор рассказов имеет по одному входному полю для каждой конкретизации роли, а на выходе выдается тип сценария, вариант и все роли сценария. Вывод данных анализатора предложений служит вводом для анализатора рассказов.

Создание парафразы подаваемого на вход рассказа включает ряд следующих шагов.

1. Лексическое представление каждого слова подается на рассмотрение лексической карте, которая с помощью семантической карты производит семантическое представление.
2. Семантические представления по очереди подаются на вход семантического анализатора. По завершении предложения представления конкретизации ролей подаются на вход анализатора рассказов.

3. Анализатор рассказов получает от анализатора предложений последовательность представлений предложений и генерирует в выходном слое полное представление всего рассказа. Экземпляры из рассказа теперь оказываются связанными с ролями.
4. Представление рассказа подается в эпизодическую память, которая выделяет (классифицирует) сценарий рассказа, вариант и ролевые связи. Представление полного рассказа (полной парафразы) определяется значениями векторов весов элементов-победителей в слоях сценариев, вариантов и связей.
5. Генератор рассказов, используя представление рассказа, создает последовательность предложений с конкретизацией ролей.
6. Генератор предложений, используя последовательности генератора рассказов, выводит слова (в их семантическом представлении).
7. Наконец, с помощью словаря семантическое представление слова превращается в его лексическую форму.

Вопрос может быть переведен в форму представлений конкретизации ролей, которые составитель подсказок использует для того, чтобы построить приближенное представление рассказа. Затем эпизодическая память получает подсказку для определения сценария, варианта и ролевых связей, используемых впоследствии для формирования ответа.

Качество работы DYNASTY и DISCERN

Обе модели показали хорошее качество обобщения. Для модели DYNASTY в [Lee *et al.*, 1990] использовались четыре сценария (посещение ресторана, прослушивание лекции, поход за покупками и визит к врачу) и были сгенерированы по восемь примеров каждого из них. Из 32 примеров сценариев, 16 использовались для обучения и столько же для тестирования качества обобщения. Все учебные и тестовые сценарии были обработаны правильно. Миккулайнен (Miikkulainen) тестировал модель DISCERN на 96 рассказах, полученных в результате конкретизации трех сценариев и трех их вариантов. В сгенерированном тексте 98% слов были правильными. Очевидно, что обе системы выполняли свою работу хорошо.

Модель DISCERN является более полной моделью по сравнению с DYNASTY хотя бы потому, например, что анализатор (в модели DYNASTY) не был реализован, а оставлен для реализации в дальнейшем. Центральным с точки зрения качества работы обеих систем является механизм формирования представлений, а здесь обе системы используют очень схожие методы рециркуляции. Представления FGREP, создающиеся в модели DISCERN, накапливаются во время обучения системы вы-

полнению задачи, тогда как распределенные семантические представления в модели DYNASTY формируются независимо от задачи. И хотя представления FGREP настраиваются именно в соответствии с решаемой задачей, распределенные семантические представления, будучи найденными независимо от конкретной задачи, должны быть переносимыми и на другие задачи, требующие доступ к словесному содержанию (см. [Lee *et al.*, 1990]). В общем, обе модели выполняют свою работу хорошо на практически одинаковых задачах, и обе предлагают хорошие примеры модульной архитектуры.

В [Miiikkulainen, 1995] описана другая модульная архитектура, названная SPEC (Subsymbolic Parser for Embedded Clauses — субсимвольный анализатор для подчиненных предложений), предназначенная для обработки предложений с подчиненными предложениями (например, “Та девушка, которой понравилась собака, запомнила хозяина собаки”). Модель SPEC разрабатывалась для анализа предложений с новыми комбинациями подчиненных структур, не предлагавшихся в процессе обучения. Центральным компонентом системы является теперь уже хорошо знакомая нам сеть SRN, которая строит отображение последовательностей в представления конкретизации ролей. Этот анализатор на основе сети SRN дополняется реализацией стека RAAM, предназначенного для хранения выражений (например, “Та девушка”), используемых в ходе анализа позже, а также сетью сегментации, являющейся сетью с прямой связью, выполняющей разделения предложения на выражения.

8.5. Другие вопросы, касающиеся представлений

8.5.1. Обобщение

“Обобщение” является термином с очень широкой областью использования. Человек выполняет обобщения без особых усилий. Например, вы узнаете лицо друга, даже если он отрастит бороду или изменит причёску. Обобщение может означать и нечто более абстрактное. Например, совсем не обязательно явно информировать о том, что объект, поставленный на колеса, будет легче передвинуть, чем объект без колес — мы можем догадаться об этом и сами. Опыт попыток перемещения объектов самых разных типов дает нам возможность вывести подходящее знание, так что когда мы видим новый объект в первый раз, можно достаточно точно оценить те усилия, которые потребует его перемещение. Знания, касающиеся преимуществ использования колес, должны как-то абстрагироваться от объекта в целом, поскольку объекты с колесами по внешнему

виду могут сильно отличаться: далеко не всегда мы, указывая на объект, говорим, что он подобен некоторому другому объекту и поэтому передвинуть его будет легко. Точно также, наличия колес не достаточно для обобщения: существует немало объектов, не предполагающих перемещение, но имеющих детали, напоминающие колеса. Абстракция знания должна делать знание переносимым. Например, инженер-разработчик знает, что тележка для супермаркета должна быть с колесами, а мы можем заключить, что стиральную машину будет легче передвинуть, если поставить ее на ролики. Знание о колесе существует в форме, допускающей применение этого знания в очень широком диапазоне сценариев.

В работе [Barnden, 1992] предлагается пример явного обобщения. Нейронные сети по большей части используют неявное обобщение. Например, сеть можно обучить связывать фамилии конкретных коммунистов, живущих в городе, с понятием “атеист”. Систему можно обучать обобщать так, чтобы для любого коммуниста, проживающего в городе, ее выходное заключение об активности включало характеристику “эта персона является атеистом”. Система при этом не строила обобщение типа “все или большинство живущих в городе коммунистов являются атеистами”. Предположим, что явным обобщением является подача на вход системы требования использовать следующее правило.

Если все (или большинство) живущих в городе коммунистов являются атеистами, то город имеет право претендовать на дотации фонда Фанди.

Пример Барндена (Barnden) выявляет разницу между обобщением знаний в случае конкретных экземпляров и обобщением на некоторый класс. Право претендовать на дотации фонда Фанди требует обобщения с использованием классов.

Предположим, что вы выросли, не зная того, что такое чашка чая, и кто-то (кто предполагает, что вы пили чай раньше) предлагает вам чашку чая, когда вы пришли к нему в гости. Через некоторое время вы уже будете знать, что чашка с красно-коричневой жилкостью, поставленная перед вами, скорее всего, будет горячей (вам также несколько раз подавали холодный чай). Вы можете сделать обобщение, заключив, что “обычно чай подается горячим”. Явное обобщение делает знание переносимым. Вы можете, например, понять ассоциации предложения “купите стеганный чехол для чайника, если вам нравится горячий чай”.

Легко понять, почему многие прагматики выступают за гибридные системы — системы, которые используют преимущества как символической парадигмы, так и парадигмы нейронных сетей. “Гибрид” как термин является отчасти не слишком удачным, поскольку для разных людей он означает разные вещи. Для одних гибридная система означает систему, со-

держашую разные модули, которые могут быть идентифицированы как символьные или основанные на нейронных сетях. Для других такой синтез может означать более сильную интеграцию в рамках архитектуры, например типа RAAM, являющейся моделью сети, предназначенной для представления символьных структур. Но мы можем игнорировать степень гибридизации, чтобы выяснить, в чем привлекательность самой идеи.

Рассмотрим контекст. Символ Ψ является буквой “v” или “u”? Распознавание символа не обязательно должно быть проблемой, поскольку мы можем принять решение по имеющемуся контексту. В показанном на рис. 8.16 случае этот символ в слове PUT интерпретируется, как “u” (нетрудно догадаться, что символ Ψ должен быть словом). Но каким является второе слово — DOVE или DOUE? Разрешить эту неоднозначность поможет дополнительный контекст (скажем, если слово является частью предложения). Для решения неоднозначности интерпретации буквы можно использовать гибридный подход: создать нейронную сеть, выбирающую несколько возможных кандидатов (например, “u” и “v”) и использующую символьные правила для кодирования слов и знание структуры предложений для окончательного выбора варианта. В данном случае нейронная сеть должна рассмотреть один символ и с помощью правил обработать контекстуальную информацию. Как вариант можно было бы решить всю задачу с помощью нейронной сети. Эта задача на самом деле кажется очень подходящей для использования нейронных сетей ввиду того, что она требует распознавания образов, что нейронные сети делают достаточно хорошо, а кроме того, мы знаем, что представление контекста тоже связано с нейронными сетями. Но тут наступает момент, когда для процессов высшего уровня требуется механизм абстрагирования знаний, а это до сих пор для коннекционистов оказывается весьма трудной проблемой.

PUT DOUE

Рис. 8.16. Символ Ψ может читаться как u или v в зависимости от контекста, в котором он встречается

Способность абстрагироваться необходима для решения сложных задач. Через абстракцию мы можем отбросить детали и сконцентрировать ресурсы на существенных элементах задачи. Проектируем ли мы здание или планируем праздничный вечер, мы сначала работаем с минимальной детализацией, чтобы держать работу под контролем, добавляя информацию постепенно. Абстракция имеет с языком фундаментальные связи. Рассмотрим следующие утверждения.

Животные имеют кожу.

Животные дышат.

Животные едят.

Эти утверждения абстрагированы на основе наблюдений за многими живыми созданиями нашего мира. Через эти утверждения знание делается переносимым для использования в самых разных контекстах. Например, вам нет необходимости знать, что такое собака, если вам скажут только, что это животное, чтобы заключить, что если у вас будет собака, вы обязаны будете ее кормить. Для вас даже не является необходимостью хотя бы раз в жизни видеть реальную собаку, чтобы знать о ней достаточно много.

Язык для людей является очень богатой формой коммуникации. Посредством языка можно с помощью инструкции моделировать образец поведения в определенном сообществе, участвовать в играх со сложными правилами и, как мы уже видели, передавать знания. Символьные системы являются языковыми системами. Все ученые, занимающиеся компьютерными разработками, хотели бы получить в свое распоряжение простые в использовании средства передачи знаний машине. Мы видим это по непрекращающимся попыткам разработки все новых и новых языков и версий языков программирования всех уровней, от уровня машинных кодов до самого высшего уровня. Мы требуем создания систем, которые было бы легче программировать. В настоящее время большинство систем требуют от человека сначала решить стоящую перед ним задачу, а затем сообщить полученное решение машине посредством языка программирования. Преимущество обучающихся машин типа машин на основе нейронных сетей состоит в том, что в них программа появляется как побочный продукт обучения машины решать поставленную перед ней задачу. Но мы не сможем использовать потенциал таких машин полностью, пока не научимся общаться с ними. И даже ограниченная форма общения (например, в терминах реакции машины на команды) была бы немалым шагом вперед. В следующем подразделе вводится понятие обоснования символов, которое может оказаться существенным для вопросов, связанных с возможностями машин проявлять истинное понимание и способность к общению. Затем следует краткое обсуждение двух систем на основе нейронных сетей, дающее пример того, как можно было бы реализовать машинное общение.

8.5.2. Проблема обоснования символов

В работе [Harnad, 1993] Харнад заявил следующее.

Познание не может быть только вычислением, поскольку вычисление представляет собой лишь систематически интерпре-

тируемую манипуляцию бессмысленными символами, тогда как содержание моих мыслей не зависит от их интерпретируемости или интерпретации кем-то другим.

Согласно Харнаду, символьное содержание должно выражаться в рамках содержания автомата.

Ранее в этой главе мы уже говорили о том, что символы являются бессмысленными сущностями в себе, которые могут выражать значение только при интерпретации их некоторым агентом, уже понимающим то, что означают эти символы. Способность воспринимать внешнюю среду и взаимодействовать со средой дает машине потенциальную возможность выработать для себя понимание объектов и событий в окружающем мире. Харнад считает, что нейронные сети являются потенциальными “кандидатами” в системы обоснования символов.

В [Harnad et al., 1994] отмечено, что люди видят объекты по-разному, когда учатся сортировать их по категориям. Объекты одной категории кажутся более похожими, а объекты разных категорий — менее. Харнад и его соавторы использовали сеть с обратным распространением ошибок, чтобы продемонстрировать создание таких категорий, где объекты имели пространственное представление в терминах значений активности скрытых элементов. Согласно авторам указанной работы, такими категориями могут быть заданные символьные метки, и эти символы могут быть объединены в строки, формирующие утверждения, касающиеся объектов. По мнению авторов указанной работы, значения таких символических представлений должны быть “обоснованы” во внутренних единицах системы, чтобы выбрать из их сенсорных прожекций объектные категории, содержащиеся в утверждениях.

Чтобы сформулировать идею обоснования яснее, мы используем более ранний пример Харнада (см. [Harnad, 1990]). Представьте себе робота, работающего в реальном мире и обучающегося выделению одной категории для объектов, которые мы называем “лошадь”, а другой категории — для объектов, которые мы называем “полоски”. Если этим категориям присвоить соответственно метки “лошадь” и “полоски”, то символы “лошадь” и “полоски” окажутся обоснованными. Затем из “зебра = лошадь & полоски” можно сформировать обоснованный символ для “зебра”.

Предложенный Харнадом подход является гибридным. Можно представить себе нейронную сеть, присоединенную к сенсорному аппарату. В результате наблюдения множества объектов реального мира в пространстве представлений сети образуется определенная область, содержащая все, что мы знаем об объектах типа “лошадь”, и другая область, где находится все известное об объектах типа “полоски”. Нейронная сеть воспринимает информацию и проводит категоризацию. Затем категории

могут быть помечены символами, и с помощью комбинирования символов могут генерироваться новые символы. Символы типа “зебра = лошадь & полосы” создаются комбинацией символов, но предполагается, что символ “зебра” должен оказаться обоснованным ввиду того, что он возникает в результате композиции представлений категорий.

В работе [Sharkey, Jackson, 1994] такой гибридный подход признан далеким от совершенства. Используем два примера из этой работы: представьте себе лошадь с тонкими полосками вдоль спины или лошадь, накрытую полосатой попоной. Назвали бы вы такие композиции зебрами? Конечно же нет, хотя оба объекта являются композициями вида “лошадь & полосы”.

Контекстная зависимость представлений нейронных сетей, которые мы часто признаем исключительно полезными, бывает источником дополнительных проблем. Рассмотрим понятие “кофе”. “Кофе” имеет представление, зависящее от контекста. “Кофе” в банке должно иметь представление, отличающееся от “кофе” в чашке (хотя эти представления и должны быть в определенном смысле подобными). Как можно идентифицировать понятие “кофе”, чтобы сделать это понятие переносимым на другие случаи использования? Символы оказываются переносимыми по причине того, что они дискретны, произвольны и независимы от контекста. Возможно, мы нуждаемся в контекстно-независимых компонентах представлений нейронных сетей? В сети с прямой связью из работы [Sharkey, Jackson, 1994] активности скрытых элементов являются контекстно-зависимыми, а весовые значения — контекстно-независимыми представлениями. Чтобы понять эту идею, рассмотрим сеть RAAM, создающую представление для “чашка кофе” и “банка кофе”. “Кофе” в этих выражениях имеет разное представление, поскольку в первом случае кофе выступает в композиции с “чашкой”, а во втором — с “банкой”. Однако для обозначения символа “кофе” во входном слое будет один или несколько элементов, и по окончании обучения весовые значения связей, соединяющих соответствующие кофе элементы со скрытым слоем, будут оставаться неизменными. Сигнал возбуждения по соответствующим связям будет посылать нечто иное, как символ “кофе”, и этот сигнал будет одним и тем же для всех экземпляров “кофе”, независимо от контекста, в котором будет встречаться “кофе”. Весовые значения, таким образом, могут служить в качестве контекстно-независимых представлений. Так что вместо обоснования символа “чашка кофе” как композиции “кофе & чашка”, при котором используются метки категорий, предлагается использовать контекстно-независимые весовые значения для “кофе” и “чашка”, с помощью которых на элементах скрытого слоя формируется контекстно-зависимая композиция для символа “чашка кофе”.

Понятие обоснования символов может оказаться исключительно важным для разработки истинно интеллектуальных систем. Возвращаясь к понятию “колесо”, кто возьмется полностью описывают все то, что следует знать об этом понятии? Где заканчивается знание о понятии “колесо”? Разве в процессе развития общества наши знания об объектах не расширяются? А если мы не можем собрать воедино все знания о колесе, то сможем ли мы тогда сделать эти знания доступными для использования в самых разных сценариях? Возможно, именно идеи Харнада вместе с идеями Шарки и Джексона и других авторов обеспечат нам движение вперед!

8.6. Возможность машинного общения

В этом разделе будут рассмотрены две системы, являющиеся примерами того, какой подход можно использовать при разработке системы общения с машиной. Здесь следует подчеркнуть, что до реализации свободного диалога с машиной еще очень далеко, но нет никаких оснований не ожидать некоторых полезных разработок уже в ближайшие несколько лет. Сейчас существуют интеллектуальные Web-агенты, принимающие команды на английском языке для поиска документов в сети. Однако пока что эти агенты представляют собой не более, чем “намек” на то, что должно быть в будущем.

Авторы работы, представляемой в этом разделе, не ставили своей целью демонстрацию возможностей машинного общения как такового, но мы можем взглянуть на их работу и с точки зрения перспективы коммуникаций.

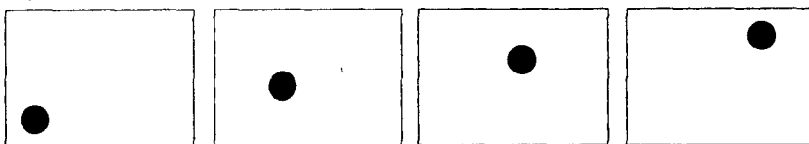
Ненов (Nenov) и Дайер (Dyer) разработали систему DETE (см. [Nenov, Dyer, 1994] или [Dyer, 1994]). Система DETE предназначена для восприятия ограниченной среды и взаимодействия с ней. Эта система оборудована глазом, с помощью которого воспринимаются пятна, перемещающиеся на визуальном экране, и пальцем, с помощью которого можно толкнуть пятно или коснуться его. В ходе обучения воспринимается три вида информации: фреймы (снимки) экрана (представляющие стационарные или движущиеся пятна), последовательности команд движения (например, для перемещения глаза) и речевые описания визуальных/двигательных последовательностей. По завершении обучения система DETE может обеспечить речевое описание визуальной/двигательной последовательности или, наоборот, генерировать последовательность визуальных/двигательных представлений по устному описанию.

Визуальные и двигательные признаки связываются со словами/выражениями. Например, команда “Коснуться большого зеленого квадрата” сигнализирует пальцу о том, чтобы он вступил во взаимодействие с объектом, значениями признаков которого будут “большой”, “зеленый” и “квадрат”. Мир системы DETE, хотя и является ограничен-

ным, достаточно богат в смысле возможных пространственно-временных ассоциаций. Например, квадрат может находиться слева от треугольного пятна, двигаться по экрану слева направо, и на какой-то стадии квадрат может “догнать” треугольник, если квадрат движется быстрее. Слово “догнать” является ассоциацией между пятнами, зависящей от изменения пространственного размещения пятен во времени.

Для представления визуальных/двигательных признаков используется девять фреймов признаков, каждый из которых состоит из двумерного массива элементов, имеющих размер 16×16 . Внутри каждого фрейма признака пятно представляется четырьмя соседствующими элементами. Для каждого конкретного пятна признак может быть меняющимся или инвариантным. Например, круг на рис. 8.17 передвигается от нижнего левого угла экрана к верхнему правому углу. Круглое пятно в данном случае остается той же формы и того же размера, и движется с одной и той же скоростью. В плоскостях признака для формы, размера и скорости перемещения для каждого временного фрейма будет активизирован один и тот же набор элементов, но по мере того, как круг движется, во фрейме признака размещения будут активизироваться разные элементы.

Экран



Плоскость признака размещения



Плоскость признака формы



Время

Рис. 8.17. Фреймы признаков системы DETE. С фреймами движения экрана связывается серия фреймов признаков, описывающих изменение этих признаков во времени

На визуальном экране одновременно могут присутствовать несколько пятен. Разным пятнам назначаются разные фазы активизации в плоскостях признаков, чтобы учесть возможность “визуального перекрещивания” (например, если большой треугольник и маленький квадрат присутствуют на экране одновременно, система без учета фаз не сможет отличить эти объекты от маленького треугольника и большого квадрата).

Для речевого ввода используется упрощенное фонематическое представление. По сути используется 64-битовая строка, каждый бит которой означает наличие или отсутствие определенного значения частоты. Фонемы повторяются так, чтобы продолжительность речевого описания соответствовала времени присутствия капли на визуальном экране. Чтобы выучить значение выражения “шар движется”, это выражение воспроизводится для разных видеопоследовательностей, в которых признакам размера, скорости и т.д. разрешено меняться, но плоскости признаков формы и движения остаются инвариантными (потому что шар всегда круглый и всегда его перемещение описывается как “шар движется”).

Система DETE включает ряд модулей памяти, каждый из которых имеет нейронную архитектуру, называемую “катамической памятью”. Катамическая память представляет собой сложную архитектуру, предназначенную для создания ассоциаций последовательностей и завершения последовательностей по подсказкам в виде части последовательности. Эта архитектура включает модули памяти для каждой из плоскостей признаков, задачей которых является кодирование записей памяти, отражающих изменяющуюся активность в плоскостях соответствующих признаков.

Система DETE обучается постепенно: сначала запоминаются отдельные слова, затем пары слов и т.д., в результате чего формируются все более длинные последовательности слов. Система DETE обладает способностью обобщения новых последовательностей слов, показанных ей в ходе обучения. Данная система обучалась также отвечать на простые вопросы с помощью завершения последовательностей. Обучение заключалось в использовании последовательностей, содержащих рассматриваемые в визуальном контексте вопросы и ответы:

“какой_размер” (визуальная сцена: большой зеленый треугольник)?, ответ “большой”:

“какой_цвет” (визуальная сцена: маленький красный шар)?, ответ “красный”.

Система DETE обучалась и последовательностям движения. Например, системе можно было показать два пятна и задать речевой ввод “что_больше”, на который она должна была ответить выбором подходящего пятна с помощью глаза и генерированием речевого ответа.

Ноэль и Коттрелл (см. [Noelle, Cottrell, 1995]) тоже исследовали ассоциацию лингвистического вывода с восприятием. Их подход был очень

простым. Архитектура сети складывалась из двух сетей SRN, как показано на рис. 8.18. Выходной слой порождает лингвистическое описание воспринимаемого события, поданного на рассмотрение слоя сцены. Слой сцены принимает последовательности простых (имеющих всего два пикселя в высоту и четыре пикселя в ширину) представлений шарика, катящегося, летающего или прыгающего на экране. За один шаг времени создается одна лингвистическая лексема, так что описание типа “шар прыгает вправо” должно порождаться за три последовательных шага. Для каждого шага времени в слой сцены подается картинка изображения. Когда требуется речевое описание, значения активности подсети изображения замораживаются, а подсеть описания проходит три рабочих цикла, требуемые для создания лингвистического вывода.

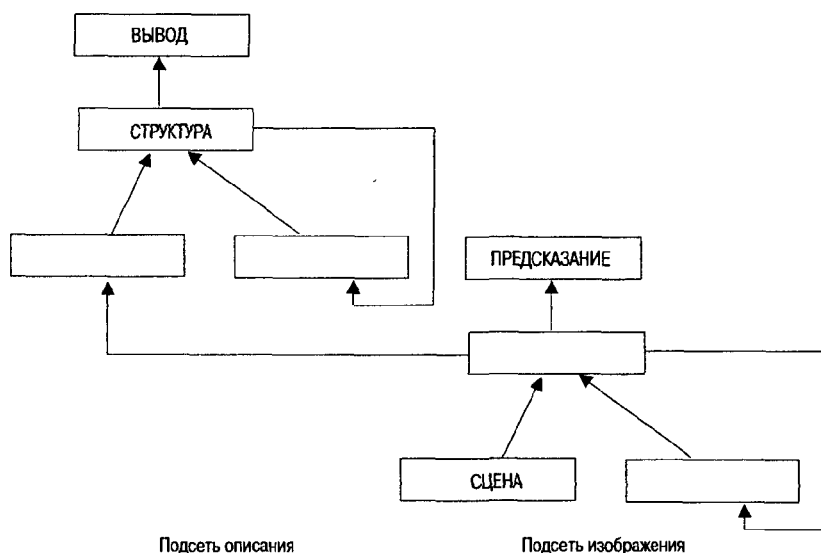


Рис. 8.18. Использование двух сетей SRN, обученных связывать лингвистический вывод с простыми последовательностями изображений

Целью Ноэля и Коттрелла была демонстрация того, что лингвистические строки могут быть внутренне связаны с некоторой специальной задачей. Это, вместе с идеей о том, что система на основе нейронной сети может обладать систематическим поведением (что продемонстрировала их сеть-сумматор, о которой говорилось в главе 5), легло в основу идеи управляемой сети, т.е. сети, которая может реагировать подходящим образом на некоторые простые последовательности команд. Сам экспери-

мент Ноэля и Коттрелла был очень простым, а решаемая специальная задача не особенно интересной, но их целью была демонстрация потенциальной пользы управляемой сети и демонстрация того, что эта польза вполне достижима. Известно несколько подходов, позволяющих внедрить в нейронную сеть априорные знания для того, чтобы начать обучение. Идея Ноэля и Коттрелла оказывается интересной потому, что она интегрирует обычное обучение нейронной сети и “обучение по инструкциям”. Успешная реализация такой идеи могла бы привести к динамически адаптируемой системе: если необходимо, чтобы система изменила свою реакцию на определенную ситуацию, системе можно просто приказать! Возможность общаться с сетью с помощью речи может быть полезной и в ходе проверки качества работы системы: если система реагирует на инструкции быстро и правильно, то заложенные в систему знания можно считать подходящими. Кроме того, как отмечают Ноэль и Коттрелл, такая система могла бы помочь исследователям в деле понимания того, как протекают процессы получения умозаключений на высшем уровне, а также того, какие более простые ассоциативные механизмы лежат в основе этих процессов. Ноэль и Коттрелл экспериментировали с целым рядом архитектур, но мы вкратце рассмотрим только одну из них.

По своей сути обучение сети реагировать на инструкции ничем не отличается от любого другого вида обучения. Можно, например, кодировать инструкции как образцы, соединяя их затем с учебными данными (т.е. связывая инструкции с экземплярами учебных образцов конкатенацией). Однако с технической точки зрения лучше хранить инструкции отдельно от вводимых данных. Также желательно иметь средства для ввода инструкций переменной длины, и это сделать легче, если инструкции и данные хранятся отдельно.

Ноэль и Коттрелл экспериментировали с дискретными отображениями. Примерами являются отображения A в B или C в A и т.п. Инструкции имеют вид последовательностей, таких как, например $\Rightarrow AB$, что означает команду для сети отобразить ввод A в вывод B . Соответствующая последовательность может быть подана на вход сети, например в виде $\Rightarrow AB \Rightarrow CC \Rightarrow BC$, что означает для сети необходимость отобразить A в B , а C и B — в C . В следующий момент инструкция может быть изменена на $\Rightarrow AA \Rightarrow BB$, что означает для сети отображение A в A , B в B , а для C остается возможность отображаться во что угодно.

Модель сети показана на рис. 8.19. Обученной сети инструкции подаются в слой советов, активность которого распространяется только до слоя плана. По завершении ввода соответствующей инструкции последовательности значения активности в слое плана замораживаются (т.е. остаются в дальнейшем фиксированными) и используются в качестве констант мо-

3. Совместное использование символьных систем и нейронных сетей.
4. Внедрение элементов нейронных сетей в символьную архитектуру.

В этой главе мы рассмотрели вопросы, относящиеся к категории 2 (причем многие из всего спектра соответствующих вопросов вообще не поднимались), и некоторые аспекты модульного подхода. В сообществе коннекционистов часто подчеркивается важность распределенных представлений, но существуют и те, кто считает, что локальные представления в процессе создания высокоинтеллектуальных машин играют не менее важную роль. То, что коннекционистам просто необходимо решить проблему обработки символьных процессов высокого уровня, сомнению не подлежит. Это должно открыть путь к композиционным возможностям и систематичности. Сейчас существует мнение, что за последнее десятилетие исследования нейронных сетей привели к завершению определенного этапа. Часто цитируемые тома МакКлелланда и его группы (см. [Rumelhart *et al.*, 1986b]) были посвящены главным образом когнитивным и биологическим моделям. Тот факт, что большинство связанных с нейронными сетями публикаций имеют прикладную направленность, свидетельствует о пользе нейронных моделей. Сегодня новый акцент получает разработка нейронных моделей для интеллектуальных процессов высшего уровня. Действительно, как отмечается в [Arbib, 1995], “нам нужно больше таких книг”.

8.8. Дополнительная литература

Детальное обсуждение модулей FGREP и системы DISCERN в целом можно найти в [Mikkilainen, 1993]. Ряд статей, посвященных системам понимания речи на основе нейронных сетей, содержится в [Reilly, Sharkey, 1993], а обзор попыток решения проблемы обработки речи на основе гибридного подхода вы найдете в [Wermter, 1995]. Имеется ряд книг, в которых рассматриваются символьные связи нейронных сетей. В этих книгах вы найдете некоторые из моделей, обсуждавшихся в данной главе. Многие главы этих книг были написаны специалистами, занимающимися активными исследованиями, поэтому там вы найдете приложения, теоретические результаты и обсуждения соответствующих философских вопросов. Книгами, о которых идет речь, являются [Dinsmore, 1992], [Honavar, Uhr, 1994], [Sun, Bookman, 1995] и [Dorffner, 1997]. В качестве источника информации по ключевым вопросам мы рекомендуем использовать [Clark, 1993].

8.9. Упражнения

1. Сеть RAAM используется для кодирования следующего дерева:

$((A\ B)(C\ B)).$

(а) Изобразите соответствующее дерево.

(б) Составьте список всех поддеревьев.

(в) Составьте список подходящих векторов для терминальных символов.

(г) Предложите исходную архитектуру сети RAAM.

2. Ответьте на вопросы упражнения 1 для следующего дерева:

$((A\ (F\ G))(B\ (C\ D)(E))).$

3. Ответьте на вопросы упражнения 1 для следующего дерева:

$(A\ ((B\ C\ D)(A\ E)\ NIL)).$

4. Для дерева, приведенного в упражнении 2, выполните следующее.

(а) Опишите способ кодирования поддеревьев этого дерева.

(б) Опишите способ декодирования поддеревьев этого дерева.

5. Ответьте на вопросы упражнения 4 для дерева из упражнения 3.

6. Даны следующие шаблоны предложений.

Шаблон предложения	Конкретизации-роли
Человек взял объект	АГЕНТ-ДЕЙСТВИЕ-ОБЪЕКТ
Объект переместился	ОБЪЕКТ-ДЕЙСТВИЕ
Человек бросил человеку предмет	АГЕНТ-ДЕЙСТВИЕ-ОБЪЕКТ-ИНСТРУМЕНТ

P1. Джон взял мяч.

P2. Джон бросил Дэвиду мяч.

(а) Составьте список структурных троек слов-понятий и предложений для сети XRAAM.

(б) Для троек, о которых идет речь в п. (а), постройте деревья, отражающие структуру их кодирования в сети XRAAM.

7. Ответьте на вопросы упражнения 6 с дополнительным предложением:

P3. Мяч переместился.

8. Рассмотрим нейронную сеть с пятью двоичными элементами. Если элемент что-то представляет, его вывод устанавливается равным 1,

а если элемент не принимает участия ни в каком представлении, его вывод будет равным 0. Сколько локальных представлений может кодировать указанная сеть и сколько распределенных?

9. Слова можно рассматривать как складывающиеся из терминальных символов (букв), выполняющих конкретные роли. Например, слово ВАР имеет В в роли 1, А — в роли 2 и Р — в роли 3. Можно считать, что чем больше слова имеют одинаковых заполнителей, играющих одинаковые роли, тем более визуально подобны слова. Например, БАР в этом смысле оказывается более подобным слову ВАР, чем слову РАБ. Объясните, как на это подобие может повлиять использование для терминальных символов произвольных двоичных векторов вместо ортогональных. Иллюстрируйте свое объяснение примерами.
10. Отличается ли сеть SRN от последовательностной сети RAAM (т.е. от сети, предназначенной для представления последовательностей)? Поясните свой ответ.

Приложение А

Немного линейной алгебры

Упорядоченная пара чисел может считаться точкой или вектором в двумерном пространстве, как показано на рис. А.1. В двумерном случае говорят, что точка является элементом $\mathbb{R}^2$, в трехмерном — элементом $\mathbb{R}^3$, а в n -мерном — элементом $\mathbb{R}^n$. Для обозначения вектора в $\mathbb{R}^n$ используется запись $x = [x_1 \ x_2 \ \dots \ x_n]$. Элементы вектора могут также разделяться запятыми. Вектор может быть записан в виде строки или в виде столбца. Например, вектор

$$x = [1 \ 3 \ 2]$$

представлен здесь в виде строки, а вектор

$$y = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}$$

представлен в виде столбца. Значения элементов и их порядок у векторов x и y являются одинаковыми, в таком случае говорят, что вектор y получен транспонированием вектора x , что записывается как $y = x^t$, а вектор x — транспонированием вектора y , что записывается как $x = y^t$.

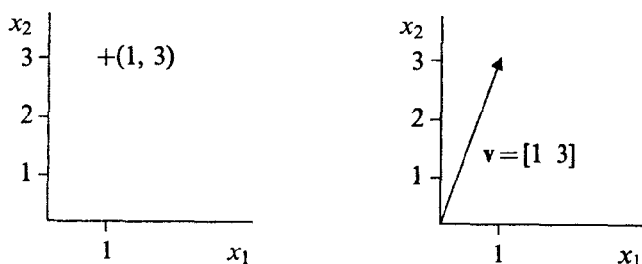


Рис. А.1. Один и тот же элемент $\mathbb{R}^2$, рассматриваемый как точка (слева) и вектор (справа)

Сложение векторов

Векторы можно складывать и вычитать. Соответствующая процедура аналогична привычным сложению и вычитанию и проводится поэлементно:

$$\begin{aligned}x &= [1 \ 3 \ 2], \\y &= [1 \ 6 \ 5], \\x + y &= [2 \ 9 \ 7], \\x - y &= [0 \ -3 \ -3].\end{aligned}$$

Умножение на скаляр

Вектор можно умножать на скаляр. Например, умножение вектора $x = [1 \ 3 \ 2]$ на 2 в результате даст вектор, который оказывается в два раза длиннее, а именно: $2x = [2 \ 6 \ 4]$.

Норма вектора

Норма или *модуль* вектора определяется по следующей формуле:

$$\|v\| = \sqrt{v_1^2 + v_2^2 + \dots + v_n^2}.$$

Так, для нормы вектора $x = [1 \ 3 \ 2]$ получаем:

$$\|x\| = \sqrt{1^2 + 3^2 + 2^2} = \sqrt{14}.$$

Скалярное произведение

Скалярным произведением двух векторов $v = [v_1 \ v_2 \ \dots \ v_n]$ и $w = [w_1 \ w_2 \ \dots \ w_n]$ является

$$v \cdot w = v_1 w_1 + v_2 w_2 + \dots + v_n w_n.$$

Также можно записать

$$v \cdot w = \|v\| \|w\| (\cos \theta),$$

где θ обозначает угол между двумя векторами.

Пример

Найти скалярное произведение векторов $[-1 \ 3 \ 6 \ -2]$ и $[1 \ 2 \ 2 \ -3]$.
Найти угол между этими векторами.

Решение

Скалярное произведение равно

$$[-1 \ 3 \ 6 \ -2] \cdot [1 \ 2 \ 2 \ -3] = -1 \times 1 + 3 \times 2 + 6 \times 2 + -2 \times -3 = 23.$$

Угол между двумя векторами равен

$$\cos \theta = \frac{23}{\sqrt{50}\sqrt{18}},$$
$$\theta = 39.94^\circ.$$

Два вектора являются *перпендикулярными* или *ортогональными*, если их скалярное произведение равно нулю. Чтобы проверить ортогональность векторов, вспомните, что если $\cos \theta = 0$, то угол θ равен 90° .

Матрицы

Матрица представляет собой прямоугольный массив чисел. Матрица размера $m \times n$ — это матрица, имеющая m строк и n столбцов. Каждый элемент матрицы можно индексировать, указав строку и столбец, в которых этот элемент размещен. Таким образом, элементы матрицы размещаются в следующем порядке:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \vdots & & & & \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} \end{bmatrix}.$$

Для определенной ниже матрицы $\mathbf{B}$ размера 3×2 имеем $b_{32} = -5$ и $b_{21} = 3$:

$$\mathbf{B} = \begin{bmatrix} -2 & 1 \\ 3 & 4 \\ 2 & -5 \end{bmatrix}.$$

Умножение матриц

Чтобы умножить две матрицы, $\mathbf{A}$ и $\mathbf{B}$ (результат записывается как $\mathbf{AB}$), число столбцов в матрице $\mathbf{A}$ должно быть равным числу строк в матрице $\mathbf{B}$. Если $\mathbf{A}$ имеет размеры $m \times n$, а $\mathbf{B}$ — размеры $n \times s$, то их

произведение AB будет матрицей размера $m \times s$. Для произведения $C = AB$ элемент этого произведения определяется как

$$c_{ij} = (i\text{-я строка вектора } A) \cdot (i\text{-й столбец вектора } B),$$

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}.$$

Пример

Вычислить произведение

$$C = \begin{bmatrix} -2 & 3 & 1 \\ 1 & 2 & 4 \end{bmatrix} \begin{bmatrix} 4 & 2 & 1 & 1 \\ -3 & -1 & 4 & 3 \\ 1 & 4 & 5 & 1 \end{bmatrix}.$$

Решение

Элементы c_{11} и c_{23} вычисляются так:

$$C = \begin{bmatrix} -2 \times 4 + 3 \times -3 + 1 \times 1 & \dots & \dots & \dots \\ \dots & \dots & 1 \times 1 + 2 \times 4 + 4 \times 5 & \dots \end{bmatrix},$$

а окончательным ответом является

$$C = \begin{bmatrix} -16 & -3 & 15 & 8 \\ 2 & 16 & 29 & 11 \end{bmatrix}.$$

Транспонирование матриц

Если матрица B получена транспонированием матрицы A , то элемент b_{ij} равен элементу a_{ji} .

Пример

Записать матрицу B , если $B = A^T$ и

$$A = \begin{bmatrix} -2 & 3 & 1 \\ 1 & 2 & 4 \end{bmatrix}.$$

Решение

$$B = \begin{bmatrix} -2 & 1 \\ 3 & 2 \\ 1 & 4 \end{bmatrix}.$$

Сложение матриц

Матрицы можно складывать, суммируя соответствующие элементы. Например, если $C = A + B$, то $c_{ij} = a_{ij} + b_{ij}$.

Представление графов с помощью матриц

Матрица может использоваться для представления графа со взвешенными путями, как показано на рис. А.2.

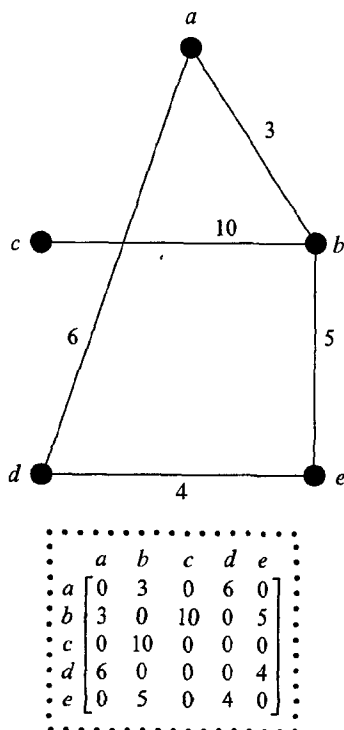


Рис. А.2. Матрица, представляющая взвешенный граф

Векторные и матричные обозначения в нейронных сетях

В резюме главы 2 приведен пример прямого и обратного проходов через сеть с прямой связью, основанный на использовании алгоритма обратного распространения ошибок. Пример представлен в форме диаграммы (см. рис. 2.22), чтобы читатель мог проверить вычисления. Для удобства та же диаграмма здесь показана снова (рис. А.3). Алгоритм обратного распространения, как и все другие сетевые алгоритмы, представлен

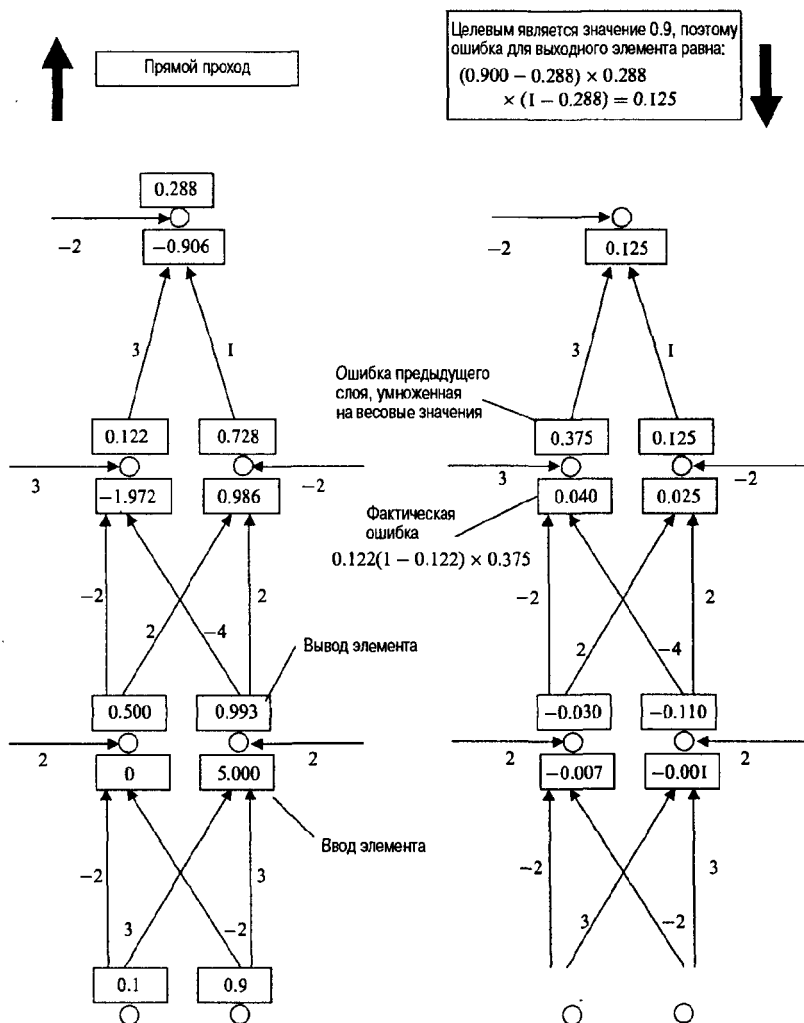


Рис. А.3. Пример прямого и обратного проходов в сети типа 2-2-2-1 с прямой связью. Данные ввoda, вывода и значения ошибок показаны в рамках (см. рис. 2.21)

в сжатом виде с помощью математических обозначений. С точки зрения вычислений в алгоритме ничего сложного нет, но описания алгоритмов могут показаться сложными, если вы не знакомы с соответствующими обозначениями. Вероятно, самым сложным делом в данном случае является сопостав-

ление индексов в уравнениях с соответствующими элементами сети и соответствующими весовыми коэффициентами.

Для случая алгоритма обратного распространения ввод конкретного элемента задается равенством

$$net_j = w_0 + \sum_{i=1}^n x_i w_{ij} ,$$

где x обозначает сигнал, посылаемый от соответствующего элемента, а w — вес связи, направленной от этого элемента к тому элементу, комбинируемый ввод которого мы вычисляем. Число элементов, посылающих сигналы, равно n . Значение w_0 задает вес смещения, который можно интерпретировать как присоединенный к элементу, чей выходной сигнал всегда равен 1. Поэтому выражение для входного значения можно переписать в виде

$$net_j = \sum_{i=0}^n x_i w_{ij} .$$

Для примера в конце главы 2 первый слой весовых значений задается матрицей

$$W_1 = \begin{bmatrix} 2 & 2 \\ -2 & 3 \\ -2 & 3 \end{bmatrix} .$$

Первый ряд задает весовые коэффициенты смещений, соответствующих первому и второму элементам первого скрытого слоя. С учетом элемента смещения, входной слой содержит три элемента, и поэтому матрица для двух скрытых элементов имеет три строки и два столбца. В скрытом слое тоже есть элемент смещения, но так как элемент смещения не может быть связан с элементами предыдущего слоя, то в матрице весовых значений этот элемент смещения не представлен.

При выполнении прямого прохода j соответствует элементу, ввод которого вычисляется, а i — элементу из предыдущего слоя. Если на вход сети подаются значения $[0.1 \ 0.9]$, то ввод для первых двух скрытых элементов задается произведением

$$[1.0 \ 0.1 \ 0.9] \begin{bmatrix} 2 & 2 \\ -2 & 3 \\ -2 & 3 \end{bmatrix} = [0 \ 5] .$$

Во входной вектор добавляется значение 1, так как именно это значение всегда посылает элемент смещения. Таким образом, комбинируемый

ввод оказывается равным 0 для первого элемента и 5 — для второго. Вычислив выходные значения для этих элементов при условии использования сигмоидальной функции активности, в выходной вектор следует добавить 1 и рассмотреть произведение полученного вектора с матрицей весовых значений второго слоя, чтобы рассчитать входные значения для элементов второго скрытого слоя. Теперь значение j должно будет соответствовать тому элементу второго скрытого слоя, ввод которого вычисляется, а i — элементу первого скрытого слоя.

В алгоритме обратного распространения ошибки скрытых элементов вычисляются в процессе выполнения обратного прохода. Ошибка любого скрытого элемента зависит от ошибок тех элементов, которым данный элемент посылает сигналы в процессе выполнения прямого прохода:

$$\delta_j = o_j(1 - o_j) \sum_{k=1}^n \delta_k w_{kj}.$$

Здесь j соответствует текущему элементу, ошибку для которого мы вычисляем, а o обозначает значение активности (выход) текущего элемента. Слой элементов, возвращающих свои ошибки, индексируется значением k . Вторым слоем весовых значений в нашем примере является

$$W_2 = \begin{bmatrix} 3 & -2 \\ -2 & 2 \\ -4 & 2 \end{bmatrix}.$$

Так как сигналы ошибок возвращаются в направлении, обратном направлению прямого прохода, то для вычисления произведения $\sum \delta_k w_{kj}$ матрицу весовых значений необходимо транспонировать. Вспомните, что индекс k соответствует элементам, посылающим ошибки обратно. Ошибки для двух элементов последнего скрытого слоя оказались равными [0.040 0.025]. Поэтому произведение, определяющее ошибки, посылаемые элементам первого скрытого слоя, будет следующим:

$$[0.040 \quad 0.025] \begin{bmatrix} -2 & -4 \\ 2 & 2 \end{bmatrix} = [-0.03 \quad -0.11].$$

Обратите внимание на то, что для обратного прохода первая строка из матрицы весов W_2 была удалена, поскольку ошибка для элемента смещения не вычисляется. Нам не требуется знать ошибки для элементов смещения, поскольку ошибки элементов влияют на изменение весовых коэффициентов, используемых в процессе выполнения прямого прохода сети, а таких весовых коэффициентов у элементов смещения, конечно же, нет, поскольку в сети нет входящих в элементы смещения связей.

Приложение Б

Словарь терминов

- Автоассоциация.** Автоассоциативное обучение используется тогда, когда необходимо запомнить образцы, которые могут впоследствии быть вызваны из памяти с помощью их искаженных помехами версий. Целью при этом является вызов истинной версии образца.
- Активность.** В контексте данной книги это сигнал, посылаемый элементом либо другому элементу, либо во внешнюю среду.
- Ассоциация.** См. *Автоассоциация, гетероассоциация.*
- Весовая матрица.** В этой матрице задаются весовые значения, характеризующие действующие в сети связи. Место весового значения в матрице определяется элементами, соединенными соответствующей связью. Иногда, чтобы описать связи элементов в сети, используют несколько весовых матриц. Например, если элементы разделены по слоям, то для описания связей между двумя соседними слоями может использоваться отдельная матрица.
- Взвешенная связь.** Связь между двумя элементами, характеризующаяся некоторым действительным числовым значением, называемым весом (а также весовым значением или весовым коэффициентом), используемым для усиления или подавления сигналов, идущих по данной связи. Элемент может быть связан с самим собой. Связь определяется начальным элементом (из которого исходит данная связь), конечным элементом (к которому связь направлена) и значением, задающим вес.
- Гетероассоциация.** Ассоциация, связывающая образец с другим образцом, а не с самим собой.

Глобальный минимум.	Состояние, при котором параметры сети (весовые коэффициенты и/или значения функции активности) являются такими, что среднеквадратичная ошибка (энергия) оказывается минимальной. Глобальный минимум представляет оптимальное решение. Локальные минимумы соответствуют минимальной ошибке (энергии) в ограниченной области параметров, но не представляют оптимальное решение.
Дуга.	Связь между двумя вершинами в представленной в виде графа структуре данных.
Инерция.	Постоянный параметр, используемый при обучении с обратным распространением ошибки. Инерция контролирует влияние предыдущего изменения весового значения на величину текущего изменения. Этот параметр помогает избежать возникновения осцилляционных изменений весов и обойти локальные минимумы.
Кластер.	Группа образцов, размещенных близко один к другому.
Композиционные структуры.	Структуры, созданные из других структур. Примером являются древовидные структуры, когда каждое дерево состоит из поддеревьев.
Коннекция.	Искусственные нейронные сети часто называются коннекциями, а парадигма нейронных сетей часто называется “коннекционизм”. Некоторые исследователи рассматривают искусственные нейронные сети как одно из средств углубленного понимания нервной системы человеческого мозга. Другие же используют термин “коннекция” для того, чтобы подчеркнуть, что нейронные сети предназначены для вычислительных целей, не связанных с достижением биологического реализма.
Локальное представление.	См. <i>Распределенное представление.</i>
Локальные минимумы.	См. <i>Глобальный минимум.</i>
Модель сети.	Термин, используемый для того, чтобы различать нейронные сети различных типов. Разные модели

	могут отличаться алгоритмами обучения, способами связывания элементов и способами обновления элементов.
Нейрон.	См. <i>Элемент</i> .
Норма обучения.	Параметр, который обычно устанавливается равным некоторому постоянному значению перед началом процесса обучения. Этот параметр ограничивает величину, на которую может измениться весовой коэффициент за одно обновление.
Обобщение.	Термин, используемый как характеристика способности сети представлять данные, на которых сеть не обучалась. Например, управляемая сеть с хорошими свойствами обобщения должна проводить правильную классификацию большинства тестовых наборов. Смысл понятия “обобщение” может слегка меняться в зависимости от контекста обсуждения, но данное здесь объяснение соответствует наиболее часто используемой интерпретации.
Образец.	Набор данных, подаваемый на вход сети. Используется также для обозначения (обучающего или тестового) экземпляра. Это может быть набор двоичных данных, описывающих изображение символа для печати, или персональные характеристики клиента, обращающегося в банк за займом.
Обучение.	Процедура, используемая для того, чтобы сеть смогла научиться выполнять поставленную перед ней задачу по предъявленным ей данным.
Обучение без управления.	Тип обучения, используемый тогда, когда нет информации о классах, на которые следует разделить учебные образцы.
Отображение.	Это понятие имеет точный математический смысл, но если говорить неформально, то отображение означает процесс, преобразующий входной образец в выходной.
Пакетное обновление.	Выполнение корректировки весовых значений после рассмотрения всех образцов. Для каждого образца происходит накопление ошибок,

	а весовые коэффициенты обновляются в конце итерации. Как правило, обновление происходит после того, как рассмотрены все образцы, но, вообще говоря, размер пакета может быть любым.
Пиксель.	Наименьший элемент изображения на экране. Чем больше число пикселей (точек), тем больше разрешающая способность.
Пошаговое обновление.	Выполнение корректировки весовых значений после рассмотрения каждого образца.
Признак.	Переменная (или атрибут). Например, признаками являются рост и вес индивидуума.
Распределенное представление.	Возникает при использовании нескольких элементов для представления одного понятия, а также одного элемента в представлении нескольких понятий. При локальном представлении каждое понятие представляется одним отдельным элементом.
Рекуррентная сеть.	Сеть, имеющая связи, по которым сигналы возвращаются обратно. Например, элемент может получать и обрабатывать сигналы от других элементов, а свой сигнал активности посылать обратно тем элементам, от которых входящие сигналы были получены.
Семантическое пространство.	Образцы, имеющие сходный смысл, должны лежать в одной плоскости пространства образов (это значит, что сходные образцы должны иметь близко расположенные соответствующие им векторы).
Сеть с прямой связью.	Сеть, все связи которой имеют одно направление от входного слоя к выходному.
Символизм.	Термин, используемый в области традиционного искусственного интеллекта для характеристики способа моделирования реального мира. Знание представляется в виде символьных структур. Например, символ может обозначать слово, представляющее одно понятие (такое как "собака"), или же может обозначать более сложный объект с набором атрибутов (имеет мех, лает, ест мясо).
Скрытые элементы.	Элементы, не связанные непосредственно с внешней средой.

Структура связей.	То, как элементы сети связаны между собой.
Топология.	В случае сетей обозначает структуру связей элементов сети.
Узел.	См. <i>Элемент</i> .
Управляемое обучение.	Тип обучения, применяемый тогда, когда известно, к какому классу относится каждый учебный образец. При управляемом обучении для каждого учебного образца известно, что для него должно быть получено на выходе сети. Если сеть не дает соответствующего результата, то алгоритм обучения использует реальный выход для того, чтобы настроить сеть (обычно с помощью корректировки весов).
Функция активности.	Функция, используемая для вычисления активности элемента по данным его ввода.
Центроид.	Усредненная характеристика размещения всех образцов в кластере. Центроид вычисляется как среднее всех соответствующих образцам векторов.
Шрифт.	Набор атрибутов, определяющих вид символов на внешнем устройстве вывода, например на экране монитора.
Элемент.	Простой процессор нейронной сети, соединенный с другими элементами взвешенными связями. В качестве других названий элемента используются термины “узел”, “нейрон” и “нейроузел”.
Эпоха.	Одна итерация всех образцов.

Список дополнительной литературы

- Ackley D.H., Hinton G.E. and Sejnowski T.J. (1985). *A learning algorithm for Boltzmann machines*. — *Cognitive Science*, (9): p. 147–169.
- Arbib M.A. (1995). Foreword in R. Sun and L.A. Bookman (eds). *Computational Architectures Integrating Neural and Symbolic Processes, A Perspective on the State of the Art*. Boston: Kluwer Academic Publishers.
- Barnden J. (1992). Connectionism, generalization, and prepositional attitudes: A catalogue of challenging issues; J. Dinsmore (ed.), *The Symbolic and Connectionist Paradigms, Closing the Gap*. Hillsdale, NJ: Erlbaum.
- Baum E.B. and Haussler D. (1989). *What size net gives valid generalization?* *Neural Computation*, (1), p. 151–160.
- Boden M. (1996). *A connectionist variation on inheritance*. Paper presented to the International Conference on Artificial Neural Networks 96, Bochum, Germany.
- Broomhead D.S. and Lowe D. (1988). *Radial basis functions, multi-variable functional interpolation and adaptive networks*. Royal Signals and Radar Establishment, Malvern. Memorandum 4148.
- Callan R. and Palmer-Brown D. (1997). *An analytical technique for fast and reliable derivation of connectionist symbol structure representations*. *Connection Science*, 9(2), p. 139–159.
- Carpenter G.A. and Grossberg S. (1987). *A massively parallel architecture for a self-organizing neural pattern recognition machine*. *Computer Vision, Graphics, and Image Processing*, 37, p. 54–115.
- Cawsey A. (1998). *The Essence of Artificial Intelligence*. Hemel Hempstead: Prentice Hall.
- Chalmers D. (1990). *Syntactic transformations on distributed representations*. *Connection Science*, 2(1/2), p. 53–62.
- Clark A. (1993). *Associative Engines. Connectionism, Concepts, and Representational Change*. Cambridge, MA: MIT Press.
- Cleeremans A. (1993). *Mechanisms of Implicit Learning, Connectionist Models of Sequence Processing*. Cambridge, MA: MIT Press.

- Copeland J. (1993). *Artificial Intelligence, A Philosophical Introduction*. Oxford: Blackwell.
- Dean T., Alien J. and Aloimonos Y. (1995). *Artificial Intelligence*. Redwood City, CA: Benjamin/Cummings Publishing Company, Inc.
- Dinsmore J. (ed.) (1992). *The Symbolic and Connectionist Paradigms, Closing the Gap*. Hillsdale, NJ: Erlbaum.
- Dorffner G. (ed.) (1997). *Neural Networks and a New Artificial Intelligence*. London: International Thomson Computer Press.
- Dyer M.G. (1994). Grounding language in perception; V. Honavar and L. Uhr (eds), *Artificial Intelligence and Neural Networks. Steps toward Principled Integration*. London: Academic Press.
- Elman J. (1990). *Finding structure in time*. Cognitive Science, 14, p. 179–211.
- Fausett L. (1994). *Fundamentals of Neural Networks. Architectures, Algorithms and Applications*. Upper Saddle River, NJ: Prentice-Hall.
- Fodor J. and Pylyshyn Z. (1988). *Connectionism and cognitive architecture: A critical analysis*. Cognition, 28, p. 3–71.
- Geman S. and Hwang C.R. (1986). *Diffusions for global optimization*. SIAM Journal of Control and Optimization, 24, p. 1031–1043.
- Harnad S. (1990). *The symbol grounding problem*. Physica D, 42, p. 335–346.
- Harnad S. (1993). *Symbol grounding is an empirical problem: Neural nets are just a candidate component*, Proceedings of the Fiftieth Annual Meeting of the Cognitive Science Society. Hillsdale, NJ: Erlbaum.
- Harnad S., Hanson S.J. and Lubin J. (1994). *Learned categorical preception in neural nets: Implications for symbol grounding*; V. Honavar and L. Uhr (eds), *Artificial Intelligence and Neural Networks. Steps toward Principled Integration*. London: Academic Press.
- Haykin S. (1994). *Neural Networks, A Comprehensive Foundation*. New York: Macmillan College Publishing Company.
- Hecht-Nielsen R. (1990). *Neurocomputing*. — Addison-Wesley Publishing Company, Inc.
- Hinton G.E., Plaut D.C. and Shallice T. (1993). *Simulating brain damage*. Scientific American, October.
- Honavar V. and Uhr L. (eds) (1994). *Artificial Intelligence and Neural Networks. Steps toward Principled Integration*. London: Academic Press.
- Hopfield J.J. (1984). *Neurons with graded response have collective computational properties like those of two-state neurons*, *Proceedings of the National Academy of Sciences*, 81, p. 3088–3092. Reprinted in J. Anderson and E. Rosenfeld (eds) (1988). *Neurocomputing: Foundations of Research*. Cambridge, MA: MIT Press, p. 579–584.

- Jordan M.I. (1989). *Serial order: A parallel, distributed processing approach*, J.L. Elman and D.E. Rumelhart (eds), *Advances in Connectionist Theory: Speech*. Hillsdale, NJ: Erlbaum.
- Kohonen T. (1990). *The self-organizing map*. *Proceedings of the IEEE*, 78(9), p. 1464–1480.
- Kosko B. (1988). *Bidirectional associative memories*. *IEEE Transactions on Systems, Man and Cybernetics*, 18, p. 49–60.
- Kremer S.C. (1995). *On the computational power of Elman-style recurrent networks*. *IEEE Transactions on Neural Networks*, 6(4), p. 1000–1004.
- Lee G., Flowers M. and Dyer M. (1990). *Learning distributed representations of conceptual knowledge and their application to script-based story processing*. *Connection Science*, 2(4), p. 313–345.
- Lippmann R.P. (1987). *An introduction to computing neural nets*. *IEEE ASSP Magazine*, 4, p. 4–22.
- Masters T. (1995). *Advanced Algorithms For Neural Networks. A C++ Source-book*. New York: Wiley.
- Miikkulainen R. (1993). *Subsymbolic Natural Language Processing, An Integrated Model of Scripts, Lexicon, and Memory*. Cambridge, MA: MIT Press.
- Miikkulainen R. (1994). *Integrated connectionist models Building AI systems on subsymbolic foundations*, in V. Honavar and L. Uhr (eds), *Artificial Intelligence and Neural Networks Steps toward Principled Integration*. London: Academic Press.
- Miikkulainen R. (1995). *Subsymbolic parsing of embedded structures*, R Sun and L A Bookman (eds), *Computational Architectures Integrating Neural and Symbolic Processes, A Perspective on the State of the Art*. Boston: Kluwer Academic Publishers.
- Miikkulainen R. and Dyer M.G. (1991). *Natural language processing with modular neural networks and distributed lexicon*. *Cognitive Science*, 15, p. 343–399.
- Nenov V.I. and Dyer M.G. (1994). *Perceptually grounded language learning*, Part 2. DETE: A neural/procedural model. *Connection Science*, 6(1).
- Niklasson L. and Sharkey N.E. (1997). *Systematicity and generalization in compositional connectionist representations*; G. Dorffner (ed), *Neural Networks and a New Artificial Intelligence*. London: International Thomson Computer Press.
- Noelle D.C. and Cottrell D.C. (1995). *Towards instructable connectionist systems*; R. Sun and L.A. Bookman (eds), *Computational Architectures Integrating Neural and Symbolic Processes, A Perspective on the State of the Art*. Kluwer Academic Publishers.
- Norris D. (1989). *How to build a connectionist idiot (savant)*. *Cognition*, 35, p. 277–291.

- Pollack J. (1990). *Recursive distributed representations*. Artificial Intelligence, 46, p. 77–105.
- Reilly R. (1992). *Connectionist technique for on-line parsing*. Network, 3, p. 37–45.
- Reilly R.G. and Sharkey N.E. (eds) (1993). *Connectionist Approaches to Natural Language Processing*. Erlbaum.
- Rumelhart D.E., Hinton G.E. and Williams R.J. (1986a). *Learning Internal Representations by Error Propagation*, in Rumelhart et al. (1986b), *Parallel Distributed Processing, Explorations in the Microstructure of Cognition*. Vol 1, Foundations. Cambridge, MA: MIT Press.
- Rumelhart D.E., McClelland J.L. and the PDP Research Group (1986b). *Parallel Distributed Processing, Explorations in the Microstructure of Cognition*. Vol 1, Foundations. Cambridge, MA: MIT Press.
- Russell S. and Norvig P. (1995). *Artificial Intelligence, A Modern Approach*. Hemel Hempstead: Prentice Hall.
- Sharkey N.E. and Sharkey A.J.C. (1992). *A modular design for connectionist parsing*, Twente Workshop on Language Technology 3: Connectionism and Natural Language Processing. Enschede, The Netherlands: Department of Computer Science, University of Twente, p. 87–96.
- Sharkey N. and Jackson S. (1994). *Three horns of the representational trilemma*, in V. Honavar and L. Uhr (eds), *Artificial Intelligence and Neural Networks*. Steps toward Principled Integration. London: Academic Press.
- Sharkey N.E. and Jackson S.A. (1995). *An internal report for connectionists*, in R. Sun and L.A. Bookman (eds), *Computational Architectures Integrating Neural and Symbolic Processes, A Perspective on the State of the Art*. Boston: Kluwer Academic Publishers.
- Specht D.F. (1990). *Probabilistic neural networks*. Neural Networks, 3, p. 109–118.
- Sun R. (1995). *An introduction On symbolic processing in neural networks*, in R. Sun and L.A. Bookman (eds), *Computational Architectures Integrating Neural and Symbolic Processes, A Perspective on the State of the Art*. Boston: Kluwer Academic Publishers.
- Sun R. and Bookman L.A. (eds) (1995). *Computational Architectures Integrating Neural and Symbolic Processes, A Perspective on the State of the Art*. Boston: Kluwer Academic Publishers.
- Werbos P.J. (1990). *Backpropagation through time: what it does and how to do it*. Proceedings of the IEEE, 78(10), p. 1550–1560.
- Weijters A., Van Den Bosch A., Van Den Herik H.J. (1997). *Behavioral aspects of combining backpropagation learning and self-organizing maps*. Connection Science, 9(3), p. 235–251.
- Wermter S. (1995) *Hybrid Connectionist Natural Language Processing*. London: Chapman & Hall.

Предметный указатель

А

автоассоциативная память, 109
автоассоциативное обучение, 124
автоассоциация, 109; 124
активность, 20
алгоритм
 детерминированный, 146
 кластеризации, 82
 обратного распространения
 ошибок, 51; 55
анализ
 семантический, 209
 синтаксический, 207

Б

бинарное дерево, 220

В

валентность, 220
вероятностная нейронная сеть. *См.*
 сеть PNN
вес связи, 15
весовая матрица, 18
весовой коэффициент, 15
взвешенная связь. *См.* связь
входной элемент, 17
высказывание, 194
выходной элемент, 17

Г

гетероассоциативная память, 109
гетероассоциация, 124
гипотеза символьных систем, 188
глобальный минимум, 147
грамматика Ребера, 135

Д

двойная импликация. *См.*
 эквивалентность
двунаправленная ассоциативная
 память, 116
двухполюсный сигмоид, 79
декодер, 220
дельта-правило, 26; 43
дерево
 бинарное, 220
 фиксированной валентности, 220
дизъюнкция, 195

З

знание, 183
 неявное, 184
 явное, 184

И

импликация, 195
искусственный интеллект, 181
исчисление
 высказываний, 194
 предикатов, 199

К

квантор
 общности, 199
 существования, 199
кластер, 80
кластеризация образцов, 80
кодер, 220
комбинированный ввод, 20
коннекционизм, 218
коннекция, 14; 218

контекстная зависимость
представлений, 258
конъюнкция, 195
корректировка весов, 41

Л

линейная проблема, 45
логистическая функция, 23
логическая связка, 194
локальное представление, 226
локальный минимум, 147; 175

М

машина Больцмана, 150
метод
Байеса, 154
инерции, 175
модельной "закалки", 146
наименьших квадратов, 28
Парцена, 154
минимизация квадрата ошибки, 43
множество противоречий, 191
модель
DISCERN, 249
DYNASTY, 243

Н

нейрон, 15
нелинейная проблема, 45
норма обучения, 27

О

обобщение, 61; 253
обучение автоассоциативное, 124
отношение XOR, 23
отрицание, 195

П

память
автоассоциативная, 109

гетероассоциативная, 109
двунаправленная
ассоциативная, 116
рекурсивная автоассоциативная, 219
перетренировка сети, 61
поиск, 188
полносвязная сеть, 51
полусумматор, 140
пороговая функция, 21
последовательность, 127
правила вывода, 197
правило
Видроу—Хоффа. См. дельта-правило
обучения, 25
предложение
сложное, 194
элементарное, 194
представление, 184
локальное, 226
распределенное, 226
проблема обоснования символов, 256
производственная система, 191
простая рекуррентная сеть. См.
сеть SRN
прототип кластера, 81
процессор, 17
прямая связь, 24

Р

рабочая память, 191
радиальная функция, 67
распределенное представление, 226
рекуррентная сеть, 109
рекурсивная автоассоциативная
память. См. сеть RAAM

С

самоорганизующаяся карта
признаков, 83
связь, 15

прямая, 24
 рекуррентная, 127
 сдвиг. См. смещение
 семантический анализ, 209
 сеть
 BAM, 116
 BP-SOM, 164
 MAXNET, 106
 PNN, 152
 RAAM, 220
 SOFM, 84
 SRN, 133
 автоассоциативная, 108
 гетероассоциативная, 108
 Кохонена. См. сеть SOFM
 модульная, 164
 полносвязная, 51
 рекуррентная, 109; 128
 с радиальными базисными функциями, 67
 Хопфилда, 109
 сигмоид. См. сигмоидальная функция
 двухполюсный, 79
 сигмоидальная функция, 22
 сила связи, 18
 символьная парадигма, 182
 синтаксический анализ, 207
 система DETE, 259
 скалярное произведение, 97
 скрытый слой, 46
 скрытый элемент, 24
 сложное предложение, 194
 слой, 17
 смещение, 22
 стратегия разрешения противоречий, 191
 структура связей, 17

Т

таблица истинности, 196
 терминальный символ, 222
 тождественная функция, 21

У

узел, 15
 устойчивое состояние сети, 112

Ф

функция
 активности, 20
 выбора решения, 38
 Гаусса, 69; 70
 логистическая, 23
 плотности распределения вероятностей, 154
 пороговая, 21
 потенциала, 154
 радиальная, 67
 сигмоидальная, 22
 тождественная, 21
 энергии, 114

Ц

центроид, 81

Э

эквивалентность, 195
 элемент, 15
 входной, 17
 выходной, 17
 скрытый, 24
 элементарное предложение, 194
 эпоха, 53

Я

ядро. См. функция потенциала
 язык Prolog, 201